



WPI

Virginia Printing

A searchable extensible online database of Virginia printing records

Project Team:

Leo Bowen-Biggs pbiggs@wpi.edu
Courtney Davis cedavis@wpi.edu

Project Advisors:

Professor Wilson Wong

Our team would like to give a special thanks to Dr. David Rawson for providing the foundational research for this project and fundamental guidance throughout the design process. By reaching out to us with his research he has made this project possible.

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see <http://wp.wpi.edu/projectbasedlearning/proven-pedagogy/project-based-learning-at-wpi>

Abstract

Maintaining a record of historical documents is critical to an accurate understanding of the present and the historical patterns that have led up to modern societal norms. The early Virginia printing trade is a critical part of the overall history of printing, and accurate historical representation of the trade is important for analyzing historical patterns. By analyzing these patterns, we can trace the progression from early printing to more modern printing practices, and eventually to the digital age of publishing. For this project our goal was to create a user-friendly website application with a database backend that will safely store and clearly present data relating to the Virginia printing trade before the 1820s. Our online application is intended for use by students, researchers, and any individual in the general public with an interest in the history of printing in Virginia from prior to 1820. The website application is intended to be maintained by historical researchers who will be able to continuously update and refine the database as new information on Virginia printing is uncovered. While our project goal was to create a website for the imprints, newspapers, and biographies we currently have on the Virginia printing trade before the 1820s, we hope that our design will allow for continual expansion of the data in the future.

Table of Contents

Abstract	i
Table of Contents	ii
Table of Figures	v
Table of Tables	ix
1. Introduction	1
2. Research	3
2.1 Index of Virginia Printing	3
2.1.1 Background	3
2.1.2 Previous Research Analysis	3
2.2 Historical Database Website Designs	4
2.2.1 Encyclopedia Virginia	5
2.2.2 Virginia Historical Society	6
2.2.3 Library of Congress Chronicling America	7
3. Software Development Methodology	8
3.1 Waterfall Software Development Paradigm	9
3.1.1 Advantages and Disadvantages of Waterfall	11
3.2 Agile Manifesto	12
3.3 Scrum	13
3.3.1 The Steps of Scrum	14
3.3.2 Advantages and Disadvantages of Scrum	15
3.3.3 Our Team's Scrum Implementation	16
4. Software Development Environment	17
4.1 Research	17
4.1.1 Communication	17
4.1.2 Repositories	17
4.1.3 IDEs	18
4.1.4 Server	19
4.1.5 Framework	21
4.1.6 Database	22
	ii

4.2 Chosen Technical Framework	23
4.3 Architecture Implementation	24
5. Software Requirements	27
5.1 Functional and Nonfunctional Requirements	27
5.1.1 Nonfunctional Requirements	27
5.1.2 Functional Requirements	28
5.2 Initial Epics and User Stories	30
5.3 Initial Use Cases	34
6. Design	45
6.1 Preliminary Design	45
6.2 Initial Architectural Design	45
6.2.1 Initial Class Diagram	46
6.2.2 Initial Entity Relationship Diagram	48
6.3 Final Design and Notable Changes	49
6.3.1 Class Diagram	50
7. Implementation	52
7.1 Original Expected Project Timeline	52
7.2 Sprint 0	53
7.3 Sprint 1	54
7.4 Sprint 2	56
7.5 Sprint 3	58
7.6 Sprint 4	60
7.7 Sprint 5	61
8. Testing	65
9. Assessment	66
10. Future Work	67
11. Conclusion	68
12. References	69
Appendix 1	71
Appendix 2	79
Appendix 3	81

Appendix 4	82
Appendix 5	83
Appendix 6	98

Table of Figures

Figure 1 - Waterfall Software Development Paradigm Steps (Price, 1999)	10
Figure 2 - SCRUM Software Development Paradigm Steps (Hughey, 2009)	14
Figure 3 - Software Architecture	24
Figure 4 - Django Architecture	26
Figure 5 - Use Case Diagram	44
Figure 6 - Initial Class Diagram	46
Figure 7 - Entity Relationship Diagram	48
Figure 8 - Final Class Diagram	50
Figure 9 - The Minimal Index page displaying links to the four test objects (Sprint 0)	54
Figure 10 - Sprint 0 Burndown Chart	54
Figure 11 - Detailed biography page for Adams, showing all basic data (Sprint 1)	56
Figure 12 - Sprint 1 Burndown Chart	56
Figure 13 - The admin interface for editing a Biography (Sprint 2)	57
Figure 14 - Chronology page displaying imprint records (Sprint 2)	58
Figure 15 - Sprint 2 Burndown Chart	58
Figure 16 - A portion of the detailed biography page showing page links (Sprint 3)	59
Figure 17 - Sprint 3 Burndown Chart	60
Figure 18 - Individual biography page for Adams, displaying all data (Sprint 4)	61
Figure 19 - Sprint 4 Burndown Chart	61
Figure 20 - The index page after the 5th Sprint	64
Figure 21 - The results displayed, next to “Refine Search” form (5th Sprint)	64
Figure 22 - Sprint 5 Burndown Chart	65

Figure 23 - The homepage of the Index of Virginia Printing	83
Figure 24 - Imprint “Typographia: An ode, on printing” available on the <i>Index</i>	84
Figure 25 - Browse imprint records by the year they were published	85
Figure 26 - Excerpt from the page for William Parks showing some of his associations	86
Figure 27 - The search box available on the Index Page	86
Figure 28 - The location of the magnifying glass in the webpage header.	87
Figure 29 - The default search results for “Adams”.	88
Figure 30 - Limiting the search for biographies who have a function listed as “editor”.	89
Figure 31 - The login page for the administrator website	90
Figure 32 - The landing page for the administrator website	91
Figure 33 - Add a Biography by clicking the “Add” button in that row	92
Figure 34 - A blank biography record allows the trusted historian to add new records.	93
Figure 35 - An error occurred trying to save this record because the ID is already in use	93
Figure 36 - The list of biographies in the database. Select one to edit.	94
Figure 37 - The form populates with the record's data and may be changed graphically	95
Figure 38 - Select multiple records to delete and choose “Delete” in the dropdown menu	96
Figure 39 - The “Recent Actions” list shows records have been added, edited, and deleted.	97
Figure 40 - Download the application code as a ZIP file from GitHub.	99
Figure 41 - Click 'cPanel' on the Client Area toolbar	100
Figure 42 - Click 'File Manager' in the cPanel toolbar.	100
Figure 43 - Select 'Upload' in the File Manager.	100
Figure 44 - The ZIP file containing the code has successfully uploaded	101

Figure 45 - With the ZIP file selected, click “Extract” from the File Manager Toolbar	101
Figure 46 - In the “Extract” dialog, leave the 'path you wish to extract the files to' blank	102
Figure 47 - Application code is available on the system	102
Figure 48 - In the initial “SSH Access” screen, click “Manage SSH Keys”.	103
Figure 49 - Under the “Manage SSH Keys” heading, click “Generate a New Key”.	103
Figure 50 - Create a strong password for the key	104
Figure 51 - The successful creation of the key	104
Figure 52 - Click “Manage” to authorize the SSH key that was created	105
Figure 53 - Click “Authorize” to authorize the SSH key for access to the account.	105
Figure 54 - Under “Databases”, select “MySQL® Databases”.	106
Figure 55 - Create a Database by giving it a name	107
Figure 56 - Create a new MySQL User with a name and strong password	107
Figure 57 - Add the user to the database that was created by selecting them from the menus	108
Figure 58 - Give the MySQL user all permissions on the database	108
Figure 59 - In MySQL Workbench, Connect to a Database	109
Figure 60 - The “Connect to Database” dialog. Fill out Hostname and Username	110
Figure 61 - Error message revealing the domain name of the local machine.	111
Figure 62 - Under the “Databases” heading, select “Remote MySQL®”.	111
Figure 63 - Add the client computer's domain name to allow it to connect to the database	112
Figure 64 - Successful connection between the client and the MySQL database	112
Figure 65 - “Select Python App” under 'Software'.	113
Figure 66 - “Setup new application” settings for the <i>Index</i>	113
Figure 67 - Settings for the new application under “existing applications”.	114

Figure 68 - Edit the WGI file location in the application settings	114
Figure 69 - Completed application settings for the Index.	115
Figure 70 - Select “Edit” to update the settings for production	116
Figure 71 - Change the Debug settings through the text editor on the server	116
Figure 72 - Create a subdomain through cPanel	117
Figure 73 - Name the subdomain 'static' and use the default document root.	117
Figure 74 - Run the ‘collectstatic’ command from the python application settings page	118
Figure 75 - Verify the collectstatic command ran properly	119
Figure 76 - Run the migrate command to configure the database tables	119
Figure 77 - The tables created by Django for the application in MySQL Workbench	120
Figure 78 - Creating a superuser on the command line	121
Figure 79 - Administrator login page	121

Table of Tables

Table 1 - Excerpt from Imprint Dataset A-G	79
Table 2 - Excerpt from Imprint Dataset H-Q	80
Table 3 - Excerpt from Journals Dataset A-G	81
Table 4 - Excerpt from People Dataset A-F	82

1. Introduction

Printing, in various forms, has been utilized for centuries to communicate ideas and preserve important records. In the past, records have been kept of these printed documents in hard copy formats, but recent efforts are being made to transfer these records from their hard copy formats into searchable online databases. The benefits of online databases of these records include being able to continuously keep up to date with new uncovered records and being able to access a multitude of records all in one convenient place. The records of the Virginia printing trade are important to gain an accurate overall view of the printing trade itself and our application will provide a location for the records to be curated. By providing a user-friendly website where the public can search through Virginia printing records, and where historians can contribute to the records, we wanted to ensure that there was a convenient and up to date online record of printing in Virginia before the 1820s. Our website utilizes a database of biographies, imprint records, newspaper lineages, and individual newspapers, all relating to printing in Virginia before the 1820s, and transforms those records into a searchable website that is easily accessible to the public.

By analyzing the data previously collected on the pre-1820s Virginia printing trade by Dr. David A. Rawson, and looking to previous historical themed sites that utilize database backends for inspiration, our team chose to follow a more relational site layout, with each data entry having connections to other data entries via page links. To create this type of website our team will follow the agile Scrum methodology, while utilizing a software stack framework consisting of MySQL, Django, and Reclaim Hosting web services. Throughout this paper we

will outline and explain, in detail, how we utilized our research and design methodology to build a historical database backed website that meets our outlined project goals and requirements.

2. Research

2.1 Index of Virginia Printing

2.1.1 Background

This project was built upon the research into the pre-1820s Virginia printing trade conducted by Dr. David A. Rawson, a bibliographer and former Worcester Polytechnic Institute history professor. The beginning stages of this project required study into the previous work of the project owner, Dr. Rawson, so that we could gain an understanding of the origin and expected goals of the project. Utilizing a written thesis and project proposal by Dr. Rawson, we were able to gain a full understanding of what was expected from us by the completion of our project, to create “a fully searchable online database....that can be readily updated as new evidence surfaces....[that follows the underlying principle of] ‘upgradeability without reconstruction’....[in order to shed light on] information that both extended the work of earlier scholars and uncovered forgotten people and imprints [relating to the print trade in Virginia pre-1820s]” (Rawson, 2016). With these goals in mind our team began breaking down the dataset of the project in order to design a well-fitting website that fulfills all the goals outlined by our project owner, but also stuck to the idea that we need to create an online user-friendly site with a well-structured database backend, in order to convey the important information relevant to the pre-1820s Virginia printing trade.

2.1.2 Previous Research Analysis

At the start of our project, our team was provided with research previously done by Dr. Rawson in the form of .pdf documents and Excel spreadsheets that pull the relevant information

from those documents (for examples of the spreadsheets we were provided see Appendices 2, 3, and 4). The research Dr. Rawson conducted resulted in the creation of three tables of data relating to the pre-1820s Virginia printing trade: imprint records, people, and journals. Each table has its own set of fields, which are explained at the end of this report (see Appendix 1). The imprint record database table contains data relating to the imprints published between 1730 and 1820, an imprint is a record created by bibliographers and cataloguers that is generated by physical examination of a printed object (Rawson, 2016). The people database table contains short backgrounds on the various individuals involved in the Virginia print trade from 1683 through 1820. Finally, the journal database table contains a history of the newspapers that were being printed in Virginia by the end of 1820, this history includes any title changes the newspaper underwent as well as the location it was printed in and who was involved in the printing. With all the provided datasets in mind our team began to research how other database backed sites linked together various records and what methods would work for our relational dataset.

2.2 Historical Database Website Designs

Efforts to centralize catalogs of historical data for online access have been successfully implemented by many other historical organizations. To gain an understanding of what our application should provide, we surveyed some of these other implementations. We specifically looked at the way they handled related data, what options were available for searching and browsing, and if they provided data related to our application. Our initial review was from a user's perspective, but we also considered how these patterns could be implemented. Overall, at the end of our research into these other sites we decided we wanted a more relational

presentation of the data, like the ‘Chronicling America’ site we explain below, to create a connective and interactive site.

2.2.1 Encyclopedia Virginia

Encyclopedia Virginia is a project by the Virginia Foundation for the Humanities and Library of Virginia. Its primary content is articles relating to a broad range of topics in Virginia History. Almost everything is an article, and there is no distinction between the type of the article's subject. For example, an article on a person and a newspaper differ only in their content, not structure or layout. They also host primary documents, which have external links to the provider of the document. These also list the articles which reference the document.

The articles have inline hyperlinks to related content. This provides an immediate access to the information but does not reveal relationships between articles in a systematic way. For example, compiling a list of people associated with a newspaper would require reading the article and finding references to those people. However, inline links provide an implicit explanation of the relationship. In the above example, reading the article provides information on how or why those people are associated with the newspaper. Certain Links within the article are highlighted on the margin. These seem to primarily point to special features (such as a “virtual tour”), or primary documents.

The website's header contains a search box. There are no immediate options available, so the search defaults to a keyword search of articles. On the results page, visitors are given the option to narrow their choice by Resource Type, Source, Medium, or Category. Alternatively, a user can browse for articles. An index provides a list of pages from A-Z, and a map provides the location of certain articles (with filters for Category and Year, and an optional search term). The

encyclopedia aims to provide many points of entry for the data by including “rich metadata associated with each entry”.

Each article page begins with a summary. Some pages have a table of contents. There are buttons, so the visitor can share the article on social media such as Facebook or Twitter. Articles also have a timeline, list of categories, list of further reading, a pre-formatted citation to use when referencing the article, a publication and modification date, and contribution credits. Additionally, each page ends in a “Give Feedback” form.

Because each page has its own URL, they can be linked to and shared externally. Encyclopedia Virginia certainly endorses other sites linking to it, but the content on the site cannot be used elsewhere as it is under copyright. Their Primary Documents have links to the owner of the document, for example the Library of Virginia (Virginia Foundation for the Humanities, 2017).

2.2.2 Virginia Historical Society

The Virginia Historical Society provides search access to their catalog of many primary documents. The results contain physical documents as well but may be limited to exclude items not available online. Advanced Search provides more filters to narrow down the search based on Keyword, Titles, Personal/Corporate Authors, Subjects, Publication Date Range, Format, or collection. However, there is no URL pattern defined so one cannot link to the results of a search. Additionally, individual records returned from a search do not have URLs either, so they cannot easily be shared or saved. Each item does have a reference number, but there is no way to call up an item based on the reference number.

The Virginia Historical Society owns the rights to the material in their collection and charges for its use. There are further restrictions on images used online, as they must “be no

larger than 72 dpi and 600 x 400 pixels”. Certain academic projects may be able to waive the fee, but they do not make finding and saving references to documents easy (Virginia Historical Society, 2017).

2.2.3 Library of Congress Chronicling America

Chronicling America is a collection of historical newspapers maintained by the Library of Congress built with sustainability in mind. The newspaper pages themselves are zoomable high-resolution scans. They also provide PDF and OCR versions. These pages link to more information about the newspaper it is from. Information about Newspapers include a standard set of metadata, much of which overlaps with the data fields required by professor Rawson. Their Search functionality is implemented through patterns in the URL, so links to search results may be shared. Additionally, each entry on a newspaper and scan of a newspaper page has a static URL to which they explicitly provide stable access. The newspapers themselves are all in the public domain, and the scanned images and webpages are provided in the public domain as well. As part of their commitment to the sustainability of the project, the source code for the website is also public domain.

Chronicling America has two types of data: Newspaper page scans and Newspaper information. They link to related information in limited and standardized ways. All newspaper page scans link to an information page on the newspaper that published it. Newspapers that are part of a lineage have preceding and/or succeeding titles linked. Any other relationships are simply collected under a “Related Links” list (Chronicling America, 2017).

3. Software Development Methodology

Before breaking down our software development methodology choices it is critical to explain key features and tools of software development. This section will help bring anyone up to speed who does not have experience with software development and may be unfamiliar with the various terms we will be using throughout the report.

For starters, what is a software development methodology, and what is it used for? Software methodologies are “frameworks used to structure, plan, and control the process of developing an information system [or other application]” (IT Knowledge portal, 2017). Methodologies are essentially the recipes for the application design that software engineers follow to ensure a successful application development. Many development methodologies utilize the same tools to accomplish setting design frameworks up. For example, both the traditional Waterfall and agile Scrum models mentioned earlier can employ use cases and user stories to convey project requirements. Use cases are descriptions of the interactions between a user and the software, while user stories are descriptions of specific needs that users have. User stories are written with the users as the audience while use cases are written using more technical language (Stellman, 2009). With the use of tools like these, frameworks can help guide software designers in the right direction of creating more user-friendly applications.

Another key concept that will be used throughout this paper is the concept of a database. While this term is probably widely known there are features that accompany database design, that we will be utilizing, that are important to explain. A database is, to put it simply, a “collection of information that can easily be managed and updated”. The data within a database is organized into rows, columns, and tables (Rouse, 2017). To display the information held in the various tables in an easy to understand way, often software developers use Entity Relationship

Diagrams. Simply put, these are diagrams that show the various relationships between database tables and what attributes each of the tables possess.

Finally, some other important terms to note are software architecture, software frameworks, and class diagrams. Software architecture is the blueprint for the application, and how all the elements that comprise an application fit together. Software frameworks are “a concrete or conceptual platform where common code with generic functionality can be selectively specialized or overridden by developers or users. Frameworks take the form of libraries, where a well-defined application program interface (API) is reusable anywhere within the software under development” (Techopedia, 2017). A class diagram is a software developer tool to show the application's classes, as well as their methods and attributes, in a convenient diagram. Class diagrams are often more technical and used for explaining coding structure. Throughout our report there will be examples of all the tools listed here that we utilized in the creation of our application.

3.1 Waterfall Software Development Paradigm

The traditional approach to software design is the Waterfall approach. The approach is very linear and implies a downward forward motion at each step, like a waterfall falling over a cliff (Hughey, 2009). In Figure 1 there are five steps outlined, the exact number of steps can vary depending on the source, but the overall idea of each phase is the same throughout sources, some just combine or break up steps for clarification and simplification.

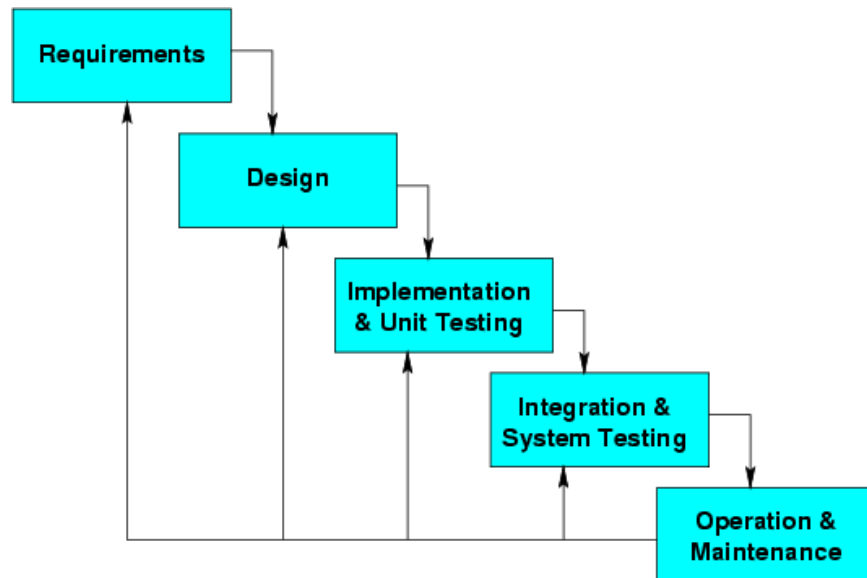


Figure 1: Waterfall Software Development Paradigm Steps (Price, 1999)

Respectively, each phase has its own steps that must be completed before a team can move on to the next one in the process. The Requirements phase is where the brainstorming of the entire project takes place. This phase is also sometimes broken up into two steps, Analysis and then Requirements, or simply combined and listed as Requirements Analysis. We chose to take the second approach in our design process, so all the requirements are outlined, and analyzed, with users during this phase. There is a large amount of time spent making sure every needed requirement is discussed and articulated and that the goals of the teams and project are all well defined by the end of this stage (Hughey, 2009). This stage requires a lot of forethought about the end goals of the project so that important requirements of the application are not overlooked. By the end of this phase, all the things an application can do should be well defined.

The design phase is first done independently of hardware or software systems. Teams take information gathered in the Requirements phase to create an abstract design of the application. The abstract design is then transformed into a concrete design that takes into

consideration the specific software and hardware requirements of the application (Hughey, 2009).

Finally, the Implementation, Verification, and Testing phases are where the application comes to life. In the Implementation phase, all the code is written, and programmers will take the requirements, specifications, and design and create a functional application. After Implementation, Verification is where the team confirms with the user and project owners that all their requirements are met by the application design. It is important in a user-oriented application design to make sure the users received what they intended to get. For the final stage, Maintenance, or 'Testing', the users use the system as they would normally and any problems that arise are changed and fixed during this phase. This is a feedback stage where the waterfall cycle has the chance to begin again if features and requirements were not met or are buggy (Hughey, 2009). The team would repeat the software development cycle to include these requirements in the application.

3.1.1 Advantages and Disadvantages of Waterfall

There are many advantages to the Waterfall design cycle and they are a big reason why this methodology has been so commonly used in the past. The first is that design flaws are caught early on, preventing a lot of implementation headaches. Also, technical documentation allows introduction of new programmers at the Maintenance phase, so they do not have to be involved in the project from the beginning to fully understand requirements. Another advantage is that the progress is very linear, and the goals are clear, so progress is easy to map. Finally, testing is easy to do as it can be done around the functional specifications laid out in the Requirements phase (Hughey, 2009).

This methodology has been used as the default for a long time, but it does carry some big negatives that have caused teams to look for alternatives, like agile methodologies such as Scrum. The first disadvantage is that users and clients are not usually able to define requirements fully in an abstract phase as many need to see an application prototype, or first design, before they can determine if it meets all the requirements they had in mind and this puts a heavy burden on the last phase of the design process as some requirements discovered missing may be hard to add afterwards. Next, the model does not cater to the possibility of requirements changing during the development cycle. This is a big problem because people often come up with ideas along the way and this design process forces all of those thought up requirements to wait until the last phase, when it may be easier to add them in earlier. Finally, the biggest drawback is that often a project takes substantially longer to come to fruition compared to if it had been designed with a different development cycle with iterations, like the agile methods (Hughey, 2009). For these reasons outlined our team considered the alternative agile methodology as we felt it may fit our project a bit better.

3.2 Agile Manifesto

To understand the more specific Scrum methodology, first there must be an overall explanation of the difference between traditional and agile methodologies. The Agile Manifesto does this in a simple and concise manner:

We are uncovering better ways of developing software by
doing it and helping others do it.
Through this work we have come to value:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on
the right, we value the items on the left more ('Agile Manifesto', 2001).

This manifesto displays the goals of the agile methodology in comparison to the goals of the traditional methodology. There is a larger emphasis on what is produced by a design methodology and dynamic accomplishment of various goals. In an agile methodology there is a great deal of focus on users, as well as application code written, to accomplish a project goal.

3.3 Scrum

The Scrum process is an agile methodology that assumes that requirements are not easy to understand early on and that they will be continuously changing and being updated, hence the term agile. There is a large focus in the Scrum methodology on user requirements and development team strategies, making it an agile type of methodology. Scrum has many features that accompany the methodology to make the design process run smoothly. These include team roles of Scrum Master, Product Owner, and Design Team, various meeting types, such as sprint planning, daily scrum, sprint review, sprint retrospective, and backlog refinement, as well as specific breakdowns of project elements, such as sprints and user stories.

3.3.1 The Steps of Scrum

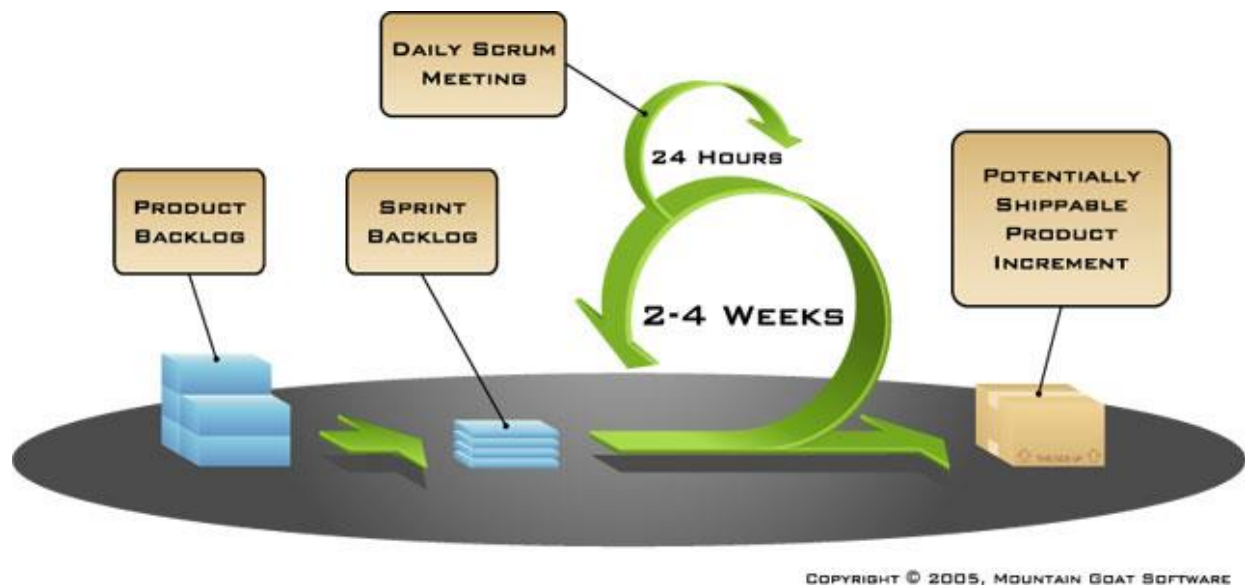


Figure 2 SCRUM Software Development Paradigm Steps (Hughey, 2009)

To understand the benefits and disadvantages of Scrum, we must first explain all the features that accompany a Scrum methodology. To do this in a way that is easy to follow we will walk through a Scrum design process, including all the features and their uses, a simple picture outlining the design process and timeline can be seen above. The Scrum methodology begins when the ‘Scrum Master’, the facilitator of all meetings, ‘Product Owner’, the owner of the specific product being designed, and the ‘Design Team’, those who do the design work, all get together and create a ‘product backlog’, or list of requirements, in the form of user stories, and even sometimes use cases. Next, the team breaks down the product backlog into multiple sprints, or iterations of design, where they pick a small set of requirements to implement, called a ‘sprint backlog’, at the ‘sprint planning meeting’. The team then goes out and performs the tasks they chose at the planning meeting and gives daily progress reports at daily meetings, or ‘daily scrum’ (Hughey, 2009). At the daily meetings, progress, new goals, and any roadblocks are discussed briefly by all team members. The next step is then to hold a ‘sprint review meeting’ where all the

work accomplished in a sprint is demonstrated to the team members. After every sprint there is also a 'sprint review meeting' where the team reflects on each other's progress. When all sprints are completed, the last meeting is held, the 'backlog refinement' meeting, where the overall sprint performance is evaluated, and the Product Owner gets a chance to review and add any requirements (Prosoft Nearshore, 2015). This is a very simplified walkthrough of a Scrum methodology format that explains the uses of all the tools involved with Scrum and the other features of a Scrum method.

3.3.2 Advantages and Disadvantages of Scrum

Scrum is a newer methodology than the traditional Waterfall model, and it has some useful advantages over the Waterfall method. Unlike the Waterfall model, Scrum does not need all requirements to be outlined at the beginning of a project, it allows, and even expects, that the requirements will be dynamic and present themselves as the project progresses. Scrum also allows for inclusion of many programming practices, like use cases, without making any type of practice go to the extreme. Finally, and probably the most defining aspect of Scrum is that it is incremental, agile, and allows for software development to be divided into iterations, or sprints, that let the teams progress enough without taking too much time for feedback and assessment (Clinger, 2016).

Like the Waterfall method, Scrum is not perfect, and it has some disadvantages that accompany its advantages. However, many of Scrum's disadvantages come from the assumed team size as, Scrum is meant for smaller teams, and when you apply it to larger groups it does not scale very well. For instance, "Scrum assumes the development team is small enough to manage itself and to fit into a small room for daily scrums" (Clinger, 2016). Also, if you are making a large system there is some difficulty in coordinating tasks and you can run into issues

where one individual's module is not compatible with another's (Clinger, 2016). Overall, if you are aware that you will be working in a smaller group, many of Scrum's disadvantages become moot points.

3.3.3 Our Team's Scrum Implementation

When evaluating our methodology options, it became clear to our team we would need an agile approach because we wanted to perform many quick and dynamic iterations, and we did not think all requirements would be apparent at the start of our design process. Our team was also a very small group, with just two members, so we knew that the Scrum method would work for us as we would not run into its disadvantages. Overall, we knew we wanted to use Scrum because of our small team size, dynamic requirements, and our desire for multiple iterations, or sprints.

In implementing the Scrum methodology, we followed the format in the following ways. Our Scrum Master was our advisor, Professor Wilson Wong and our Product Owner was Dr. David Rawson, who compiled all the research and goals for the project. We held weekly Scrums with all team members, as well as Daily Scrums over email between the two group members, as this was most convenient. We followed the other meeting patterns the way they are laid out in the Steps of Scrum section. Our team utilized user stories to define requirements for the Product Backlog and Sprint Backlogs and those can be found in our Requirements and Design sections later in this report. Overall, Scrum fit to our team dynamic and is the methodology we followed to design and implement our database and website application.

4. Software Development Environment

4.1 Research

During the process of picking the tools we would use throughout the project we had to keep in mind our team size and project requirements that we will define below in the Requirements section of this report. Our project required us to research and choose tools for communication, code storage and file sharing repositories, an Integrated Development Environment (IDE), a web server service, a stack framework, and a database.

4.1.1 Communication

For communication we had a very easy time determining what would work for our team because it is so small. Our options came down to Facebook Messenger, which is used for communication between teammates directly and quickly, as well as email, which could be used for communication between entire team, including owners and advisors, because it is a formal messaging system. We also considered, and ultimately chose, to use email for daily Scrum meetings between the two members because of its formality. Finally, we considered using SMS because of the group size and because it is an informal messaging system, but it is only available for mobile phones and is less convenient than using Messenger, which can be accessed on mobile and desktop applications. Overall, we chose to utilize all the communications we researched, except SMS because Messenger was more convenient.

4.1.2 Repositories

When we were beginning the project, we knew we wanted continuous synchronization between teammate documents and for this we considered Google Drive. Drive is used for

documents and one of its benefits is that it allows simultaneous editing by multiple people. Both team members had previously used Drive and were comfortable with its format and found it convenient, so we opted for its use to store important project documents, such as the Proposal and Meeting Minutes. Another type of repository we had to consider was for coding and for this we stuck with, to put it simply, “what we knew best”, and that was GitHub. Both members had already had existing accounts and had previous experience with how it handles version control features, as well as experience with its built-in issue tracker that we could use for assigning tasks in our sprint backlogs.

4.1.3 IDEs

Another straightforward choice for our team was what IDE to use in conjunction with GitHub. We knew we would be using Python to code our framework and we were familiar with PyCharm, a JetBrains product that supports Django, our considered web framework, development, making it an obvious choice for our team’s framework coding. We also looked at WebStorm, another JetBrains product, for front-end web development and because we could use it for developing user-facing web pages and CSS it made it an easy choice for front-end IDE. Finally, we were considering MySQL for our database, and chose it very early on, and we discovered that MySQL Workbench, MySQL’s recommended IDE, gave us access to MySQL databases in a graphical view, which we figured would be convenient for implementing our database. Overall, the decisions we made for Communication, Repositories, and IDE were very straightforward in comparison with our choices for Server, Framework, and Database, which we will discuss next.

4.1.4 Server

From the start of our project we knew that the website developed for this project would have to be hosted on a server to be publicly available on the Internet. The options we looked at for web hosting were Dr. Rawson's existing domain with Reclaim Hosting, Heroku, or a Do-It-Yourself (DIY) Approach hosted on a WPI-owned computer. We considered what languages and software they offered, how transfer of ownership might work, and pricing. Because we expected to transfer ownership of the website to a non-technical organization, we also considered how they would use the server. We found that Reclaim Hosting offered the best interface for a transfer of ownership to a non-technical organization. Also, because it was already in use by Dr. Rawson we considered it the best option to host the web application. Reclaim Hosting offers Shared Hosting for \$30 per year (Individual Plan) and Dr. Rawson was already using this service to host an interim website and to register the web domain 'indexvirginiaprinting.org'. We also discovered Reclaim Hosting is primarily for educators and students, and is based in Virginia, which are both facts that make it a nice fit for this project. Reclaim provides online browser-based access to an administrative website where website owners can manage their site. They also have many premade applications (such as Wordpress) that are ready to be installed and run, or an administrator can upload their own application to run. Additionally, Reclaim Hosting provides database hosting. This means we could configure a MySQL database on this server instead of needing to find a separate space to host our database. They also provide email hosting and allow subdomains on domains. However, with Reclaim Hosting, we must manually configure our application. There are many steps to this, such as uploading the source to the website, configuring the application, and configuring the database. The administrative portal runs cPanel, an open-source administration tool. This tool is not specific to Reclaim Hosting, so a new owner

could theoretically move the website to a new host and configure it in a similar way. Finally, Reclaim Hosting handles updates to the underlying server and database software so that the website administrator does not need to manage them.

The next server we considered was Heroku, a cloud hosting platform built mainly for developers to host web applications. Their pricing model is based on the resources used by the application. It supports many languages, including PHP, Python, Java, and JavaScript (with Node.js). They also offer Heroku Postgres which could be used to host a database. Heroku is primarily targeted at developers, which is not the target owner of our application in the future. For example, they require use of git (or another version control system) to deploy an application, which can be too technical for those with no development background. They also have integrations with GitHub to automatically deploy code or for testing new code patches, and because our team is small (only two people) and the project scope not too complex, most of those tools we felt were unnecessary. Additionally, it would be an inappropriate platform to hand off to a non-technical organization.

Finally, another approach would be to start from scratch. It is possible that we could have transferred the application to an organization that is already hosting their own website and wants to add it to their existing deployment space. However, not all web hosts support interactive web applications, so the existing infrastructure may not be appropriate. To host a website from the ground up, we would have needed to find a server at WPI willing to let us run an application and database. This option was the most complex and requires background in website administration and server maintenance. However, self-hosting provides the most low-level and host-agnostic approach and so could be appropriate to hand off to another organization. But because of the

complexity of this approach and the relative complexity of the website administration field, this is an inappropriate option for this project.

4.1.5 Framework

Broadly, we had two options: choose a traditional language and build the website from scratch or choose a full-stack framework from which to start. We looked at building a PHP or JavaScript web site or starting with the Python/Django or Ruby/Rails framework. We evaluated these based on our current knowledge of the language, the features provided, databases supported, where we could host them, and limitations imposed. We found that Python/Django provides many useful features out of the box and supports our top choice for database (MySQL).

One of the most widely used ways of building websites is with PHP (“PHP: Hypertext Preprocessor”), a scripting language primarily built and used for web development. It is supported by most web hosting solutions, including Reclaim Hosting and Heroku. Reclaim Hosting provides configuration of PHP scripts as part of their architecture. They run on the common LAMP deployment stack: Linux/Apache/MySQL/PHP. However, none of the team members have worked with PHP before and learning a new scripting language would be a huge hurdle for this project.

JavaScript was another option and is widely used on both the front end and server side of websites. On the server-side, node.js is a runtime environment that runs JavaScript code and can produce dynamic content to serve on a website. With Node.js, only one language needs to be used throughout the website as JavaScript can be used in both the front and back end. Node.js is supported by Heroku, but not Reclaim Hosting, which we felt may be a problem since we were leaning towards Reclaim Hosting.

Another choice was Ruby on Rails, a web framework that uses the Ruby programming language. It follows “convention over configuration”, which means it has implicit defaults that programmers do not need to specify. It is also “opinionated software” in that it encourages one “best” way to do things. These limitations make getting started easier because there are fewer choices to make but make it much harder down the road to do anything differently. Rails defaults to using SQLite as a Database but also supports MySQL and PostgreSQL (‘Rails’, 2017). Like PHP, neither developer on the team was familiar with Ruby on Rails, so it was not a good candidate for this project.

Finally, our last option was to utilize Python/Django: and the LAMPy Stack. Python can be an alternative to the “P” in the LAMP Stack, making it Linux/Apache/MySQL/Python, alternatively called “LAMPy”. Django is a web framework that is built with Python, which was familiar to both team members. It provides many features commonly found on websites out of the box. These include an Object-Relational Model (ORM), Admin Interface, and HTML Templating Language. The admin interface is a huge advantage as it provides a way for an authenticated user to log into the application and add, delete, or edit objects in the database. Django uses SQLite as a database backend by default, but can be configured to use PostgreSQL, Oracle or MySQL. All these options made it a choice that was high on our list, and to top it off we discovered that it was compatible with another possible choice for us, Reclaim Hosting.

4.1.6 Database

This was one of our simpler choices in architecture as well, the data we have conforms to a relational model, so a simple database like MySQL is sufficient. SQLite is a serverless database which is the default option in many frameworks and is easy to work with, however, it is generally unsuitable for production as SQLite limits the administration of the database in a

production setting. Also, SQLite does a poor job handling multiple individuals editing a database at once, which is not a good fit for this project because having many administrators was one of our intended features. For these simple reasons we figured out MySQL would be our database early in the project.

4.2 Chosen Technical Framework

We considered the various options available that we described above and how they interacted with each other when making our decision. The website we decided ultimately would be a Django application with a MySQL database running on Reclaim Hosting.

To summarize our original reasoning, we chose Reclaim Hosting because it was familiar to Dr. Rawson and most appropriate for a general audience. We chose to use a web framework to reduce the work for us and for that framework we chose Django because it supports the most configuration settings, does the most abstraction of fine details, and provides a built-in admin interface. Django also provides an Object-Relational Mapping to the database when database objects (“models”) are defined in Python. This automatically generates the SQL need to create tables and manipulate data. The only drawback to our choice we faced was that to deploy Django on Reclaim Hosting, we had to use cPanel to configure a Python Application with Django as a dependency. We then uploaded the code and configured a route for static files on Apache. However, this was not a huge overall drawback in the larger scheme as it ended up working quite smoothly. Finally, we chose to use MySQL because it is a widely-supported relational database suitable for deployment on a server. To really explain our architecture, we have broken it down in the following section.

4.3 Architecture Implementation

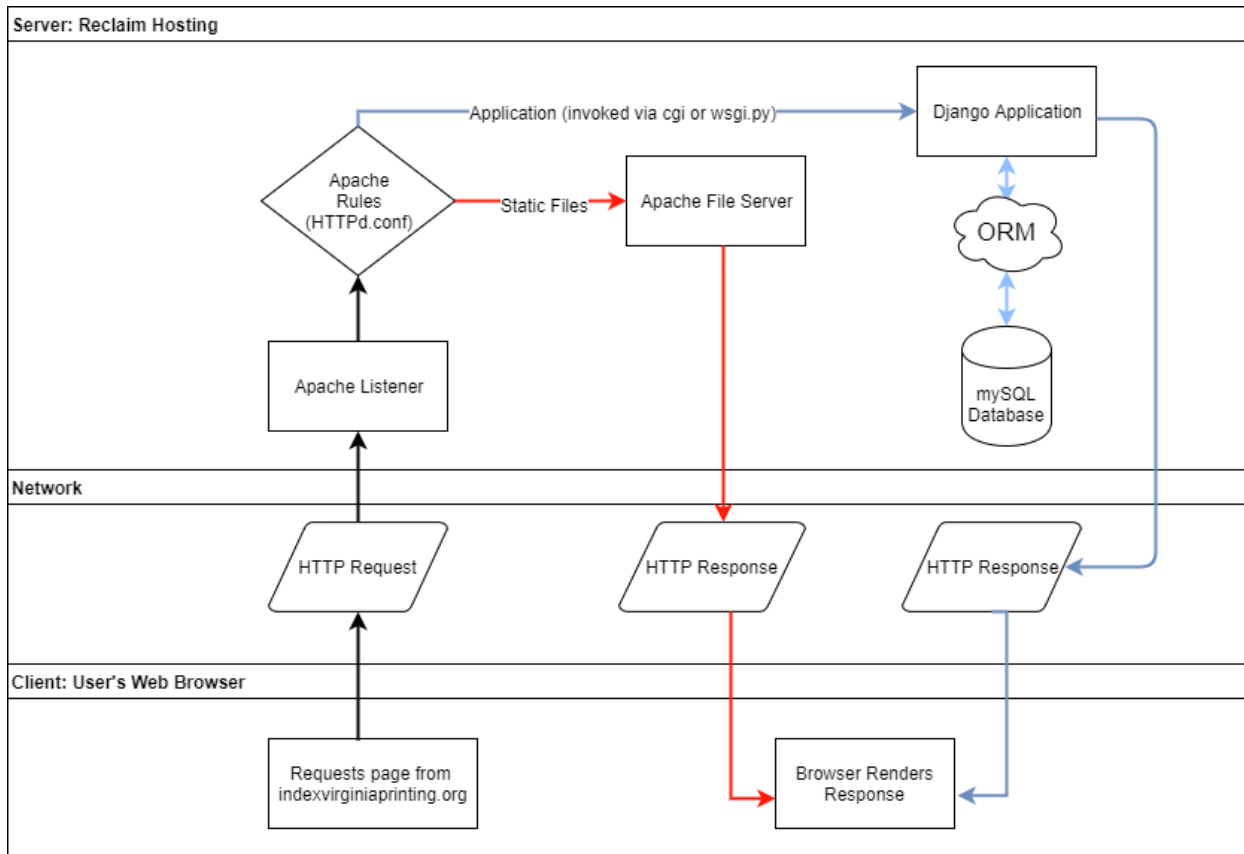


Figure 3: Software Architecture

To understand how the architecture fits together we will walk through the above architecture diagram. First, consider a request made on the website at `indexvirginiaprinting.org`. This HTTP request originates from the visitor's web browser and goes through the network to our Server at Reclaim Hosting. The Apache listener provided by Reclaim Hosting has a set of rules that determine how the request should be handled. If it is a static file, the provided Apache server simply sends an HTTP response back to the client which is rendered in the visitor's browser. However, if the requested page is part of our application, it invokes Django (via the file we specify as our application's interface, or `wsgi.py`). The Django Application then takes the request and matches the URL with a view function (via the URL Dispatcher). This is where we

determine the set of rules for how a URL translates to the page a user gets. The view function takes a template and renders it. The template may also be filled with data from the database. In this case, it calls methods on Python Objects which is translated to SQL based on the Object Relation Model. These Python Objects become the Dynamic Page Content, or “Context” that is sent with the template to the renderer. The renderer fills in the template with the data and generates a filled template in the form of a HTML document. This is then sent back to the client in an HTTP Response that is outputted to the user display.

To also show visually how the Django framework interacts with the architecture, we broke that process we just described above down into the subsection shown in the figure below. With this diagram we can visually trace how a call to the database is made by Django, where the object relational model is used, as well as where exactly database versus static calls will occur.

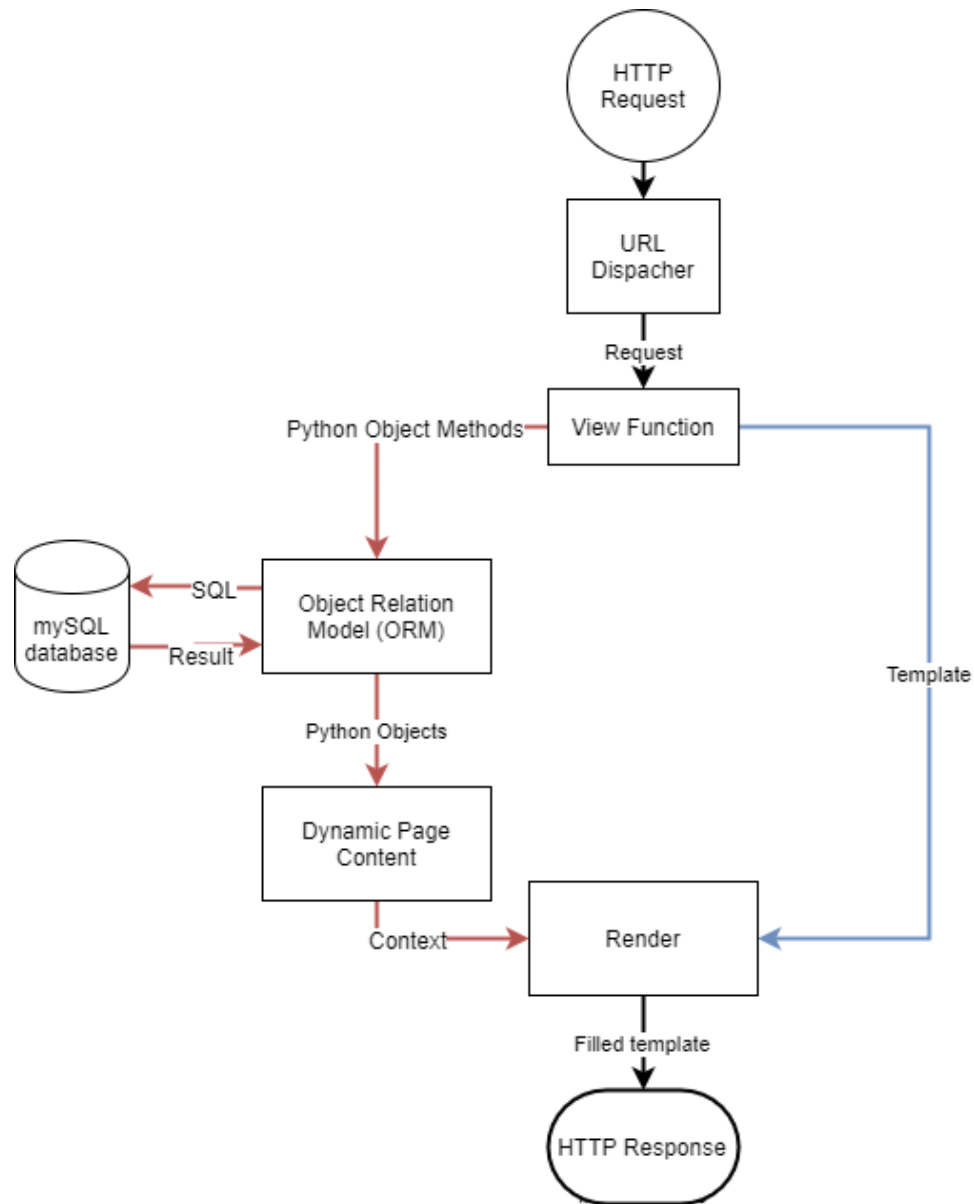


Figure 4: Django Architecture

The pieces of this architecture we need to be primarily concerned about are the Object Relational Model (ORM) Definition, the URL dispatcher rules, and the HTML templates. The ORM is highly important in our architecture because it allows us to define the data in the database and access the data without having to make direct queries, or searches, to our database from our web server. The ORM in Django will essentially act as a gateway for us between our database and web page. The URL dispatcher rules allowed us to define our URLs the way that

we liked and create a URL structure that fit our webpage and that allowed the HTML templates to create pages that would flow together and have a sensible structure. Using these framework and architecture choices and interactions, our team then laid out and finalized requirements for the first sprint design that can be seen in the sections below. The goal of the first sprint was to put this architecture into place to allow for easier expansion of the database and site content.

5. Software Requirements

5.1 Functional and Nonfunctional Requirements

Our application's requirements can be broken down into functional and nonfunctional categories. Functional requirements are *what* users wish to do with our application, while nonfunctional requirements are the system attributes our application possesses (Eriksson, 2015). Both sets of requirements define our application and what the intended production product will be capable of. We evaluated our goals outlined in this section near the end of our report in our overall assessment.

5.1.1 Nonfunctional Requirements

The nonfunctional requirements that our application must meet are longevity, scalability, usability, and security. In our later assessment of these requirements we will refer to the projected metrics defined here to determine success in meeting them.

Longevity: Our product owner would like to keep this application available for future use by other individuals. To ensure this, we must design the application in a way that it can be transferred between administrators without losing any data or functionality. The overall intended

length of use for this application after our final iteration, with only minor updating, is 10 to 15 years.

Scalability: Historical documents and databases are always being uncovered and expanded upon, so we must ensure that our application will support being updated continuously as more information and data is uncovered. We hope that our site will be able to handle at least 5,000 records, but ideally up to 10,000 records as well. That amount should allow for extensive scalability in the future.

Usability: The application must be able to support administrative users, as well as users from the general public. The administrative application is intended primarily for individuals of a historical background, and for this reason it must not be too technically complicated, as we cannot assume a technical background. The same applies for the application's general users, while there may be some technically savvy individuals that use it, technical literacy cannot be assumed. We will consider usability successful if individuals, other than ourselves, can use the site to locate a record on their own.

Security: While we are not dealing with sensitive information, it is still important to keep security in mind while dealing with databases to prevent any data loss. Our application should ensure that the database is not write-accessible to the public and that the general site is secure and not openly vulnerable to attack. This requirement will be considered successfully met if the site does not allow unauthorized users to edit pages and if it does not allow unauthorized access to administrator accounts.

5.1.2 Functional Requirements

The functional requirements that our application must meet are broken into categories relating to: storage of data, retrieval of data, entering of data, account authorizations,

administration of site, and cross-reference ability. Our requirement analysis will refer to three different types of user roles: superuser, trusted historian, and general user. The superuser is a user of the application with website formatting control and various other administrative capabilities. The trusted historian is an administrator of the site itself with ability to edit the dataset. The general user does not have any administrative or editing capabilities but can view the site application. In the analysis, roles of superuser and trusted historian may overlap and will be referred to as administrators.

Store Data: The application will be able to store data on Virginia printing in a database server that can later be accessed.

Retrieve Data: The application will be able to retrieve saved data from the database server for displaying on the site or editing by an administrator.

Account Authorizations: The application will have three user types with three authorization levels. The superuser will have the most authorization with ability to make direct changes to the site structure, content, users, and location with superuser privileges. The trusted historians will have editing privileges to the site database and record pages, but no ability to alter entire site characteristics. The general users will have read-only authorization to the site application. The application will allow both the superuser and trusted historians to log into the site to access administrative functions.

Site Administration: The application will allow those who have administrative abilities to log in and have proper access to modify the database in various ways given their respective administration levels. For example, the application would allow a super user to login and edit site appearance or location of the database, while a trusted historian would be able to login and make edits to the database entries.

Help Screens & User Documentation: The application will show helpful tips and instructions on how to use the site at the request of a user.

Cross-reference Ability: Our product owner wished to have this application connect records to one another based on their shared relationships. For example, a newspaper with many editors would contain links to the pages about those editors. Another goal of our application in terms of cross-referencing was to connect records to external sources, so that a user could find the original source of a record with ease. Historical data is all part of a huge context and the end goal of our application is that it will be able to cross reference itself and other databases to relate records and pages.

5.2 Initial Epics and User Stories

1. As an administrator, the application will allow me to safely store records relating to the Virginia printing trade before 1820.
 - a. As a superuser, I want the application to allow me to back up the database to ensure data longevity and security.
 - b. As a superuser, I want the application to allow me to save data records on a remote server to ensure data longevity.
2. As a general user, the application will allow me to search records relating to the Virginia printing trade before 1820.
 - a. As a general user, I want to be able to search for a record by a specific field in order to easily use the site and easily access specific records.
 - b. As a general user, I want to be able to search for a record by more than one criteria in order to easily access certain records.

- c. As a general user, I want to be able to sort search results by a field in order to make it easier to find a specific record I am searching for.
- 3. As a general user, the application will allow me to browse records relating to the Virginia printing trade before 1820.
 - a. As a general user, I want to browse by different data patterns in order to quickly find relevant records.
 - b. As a general user, I want to browse records in chronological order in order to gather an understanding of the data and what it represents.
 - c. As a general user, I want to view records on a map of where they were published in order to gather a deeper understanding of the dataset.
 - d. As a general user, I want to view a newspaper lineage on a timeline in order to understand the chronology of the newspaper dataset.
 - e. As a general user, I want to view help tips on how to browse the site in order to use the site to its full potential.
- 4. As a general user, the application will allow me to view records relating to the Virginia printing trade before 1820.
 - a. As a general user, I want the application to show me where to find an original version of an imprint in digital form so that I can learn more about the record.
 - b. As a general user, I want the application to save a permanent URL for a record so that I can return to where I left off in research or share my findings with others.
 - c. As a general user, I want to be able to follow a record to associated records in order to expand my knowledge on a record and find more relevant history.

- d. As a general user, I want to be able to cite a record I have found in a standard format so that I can accurately share and reference it to other individuals.
 - e. As a general user, I want to be able to use accessibility tools on the website in order to browse as easily as every other user.
5. As an administrator, the application will allow me to edit records.
- a. As an administrator, I want to add an imprint to the database application in order to expand the applications knowledge base and keep it up to date.
 - b. As an administrator, I want to add a biography to the database application in order to expand the applications knowledge base and keep it up to date.
 - c. As an administrator, I want to add a newspaper lineage to the database application in order to expand the applications knowledge base and keep it up to date.
 - d. As an administrator, I want to add a new type of record to the application in order to expand the applications knowledge base and keep it relevant.
 - e. As an administrator, I want to remove a record, regardless of type, in order to keep the collection accurate.
 - f. As an administrator, I want to flag a record or field as inaccurate, regardless of type, in order to keep the collection accurate and up to date.
 - g. As an administrator, I want to edit existing imprints in order to keep the collection accurate and up to date.
 - h. As an administrator, I want to edit existing newspaper lineages in order to keep the application accurate and up to date.
 - i. As an administrator, I want to edit existing biographies in order to keep the application accurate and up to date.

6. As a superuser, the application will allow me to manage user accounts.
 - a. As a superuser, I want to be able to add a new superuser in order to allow the site application to change ownership.
 - b. As a superuser, I want to be able to remove a superuser in order to allow the site application to change ownership.
 - c. As a superuser, I want to be able to update a superuser in order to allow the administrator details to stay accurate.
 - d. As a superuser, I want to be able to add a new trusted historian in order to allow the site application to have up to date curators.
 - e. As a superuser, I want to be able to remove a trusted historian in order to allow the site to have up to date curators.
 - f. As a superuser, I want to be able to update a trusted historian in order to allow the site to have up to date curator information.

7. As a superuser, the application will provide administrative capabilities.
 - a. As a superuser, I want to install the application on a server in order for it to be accessed by other administrators and the general public.
 - b. As a superuser, I want to migrate the application to a new server and preserve the database in order to move the site location in case of ownership change.
 - c. As a superuser, I want to back up the database in case something happens to my data.
 - d. As a superuser, I want to roll back the database to an earlier version where my data is not corrupted.

5.3 Initial Use Cases

Use Case #1

1. Name: Add a Record to the Database
2. Participating Actors: Administrator
3. Entry Condition:
 - a. The administrator is logged into the site already.
 - b. The record being added to the database is not already present.
4. Exit Condition:
 - a. The new record is present in the database.
 - b. The addition is recorded in an activity log.
5. Flow of Events:
 - a. Administrator chooses what kind of record to add.
 - b. Administrator requests to add their chosen record type to the database.
 - c. The application asks the administrator to describe the record, based on its type.
 - d. Administrator submits the information to the application.
 - e. Application saves the information.
 - f. Application records the action in an activity log.
 - g. Application shows a confirmation to the administrator.
6. Alternate Flow:

[Cancel]

1. Administrator chooses to stop at an intermediate step.
2. The application discards any information entered.
3. The application returns to the previous activity screen.

[Duplicate Found]

1. The application finds a duplicate record based on the information entered by the Administrator
2. The application warns the Administrator they are creating a duplicate.
3. The application throws out the duplicate entry and it is not added to the database.

Use Case #2

1. Name: Edit a Record
2. Participating Actors: Administrator
3. Entry Condition:
 - a. The Administrator is currently viewing the record they want to edit.
 - b. The Administrator is logged in.
4. Exit Condition:
 - a. The record is updated in the database with new information.
 - b. The edit is recorded in an activity log.
5. Flow of Events:
 - a. Administrator requests to edit the record they are viewing.
 - b. The Application asks the administrator what they want to change.
 - c. Administrator submits the information to the application.
 - d. Application saves the information.
 - e. Application records the action in an activity log.
 - f. Application shows a confirmation to the administrator.
6. Alternate Flow:

[Cancel]

1. Administrator chooses to stop at an intermediate step.
2. The application discards any information entered.
3. The application returns to the previous activity screen.

Use Case #3

1. Name: Login
2. Participating Actors: Administrator
3. Entry Condition:
 - a. The Administrator has a verified account for the application with valid credentials.
 - b. The Administrator is not already logged in.
4. Exit Condition:
 - a. The Administrator is logged in.
 - b. The login is logged in the activity log.
5. Flow of Events:
 - a. Administrator requests to log onto the system using valid credentials.
 - b. The application logs the Administrator in and provides them access to administrative functions.
 - c. The application records the login attempt in the activity log.
6. Alternate Flow:

[Invalid Credentials]

1. Administrator credentials are not registered with the system.
2. The application cancels the login attempt and the Administrator is not logged in.
3. The application records the login attempt in the activity log.

Use Case #4

1. Name: Basic Record Search
2. Participating Actors: General User
3. Entry Condition:
 - a. None.
4. Exit Condition:
 - a. The results of the search are presented to the user.
5. Flow of Events:
 - a. The user requests the application to search the database given a search term.
 - b. The application searches the database for all records matching the search term.
 - c. The application returns the results of the search to the user.
6. Alternate Flow:

[Cancel]

1. The user requests to cancel search at intermediate step.
2. The application does not return search results and returns to previous activity screen.

[No Results]

1. The application cannot find any records matching the search term.
2. The application does not return search results and instead returns a message to the user about no matching records.

Use Case #5

1. Name: Advanced Record Search

2. Participating Actors: General User

3. Entry Condition:

a. None.

4. Exit Condition:

a. The results of the advanced search are presented to the user.

5. Flow of Events:

a. The application provides the user with advanced search terms.

b. The user requests the application to search the database given selected advanced search terms.

c. The application searches the database for all records matching the search terms.

d. The application returns the results of the search to the user.

6. Alternate Flow:

[Search First Then Advanced Search]

1. The user requests the application to search the database given search term.

2. The application searches the database for all records matching the search term.

3. The application returns the results of the search to the user.

4. The application presents the user with advanced search options.

5. The user selects advanced search options.

6. The application performs the search again with the advanced options.

7. The application returns the results of the search to the user.

[Cancel]

1. The user requests to cancel search at intermediate step.

2. The application does not return search results and returns to previous activity screen.

[No Results]

1. The application cannot find any records matching the search term.
2. The application does not return search results and instead returns a message to the user about no matching records.

Use Case #6

1. Name: Browsing Records
2. Participating Actors: General User
3. Entry Condition:
 - a. None.
4. Exit Condition:
 - a. The records are displayed to the user in browsable format.
5. Flow of Events:
 - a. The application presents patterns of browsing to the user.
 - b. The user requests to browse the records in specified pattern.
 - c. The application presents the records to the user to be browsed in the correct specified pattern
6. Alternate Flow:

[Return Home]

1. The user requests to stop browsing and return to the main site page.
2. The application returns the user to the main site page.

Use Case #7

1. Name: Requesting Help Documentation
2. Participating Actors: General User
3. Entry Condition:
 - a. The user is on a page that offers help documentation.
4. Exit Condition:
 - a. The user is provided help documentation.
5. Flow of Events:
 - a. The user requests help documentation on a page.
 - b. The application presents the help documentation for the page to the user.
6. Alternate Flow:

[Cancel]

1. The user requests to no longer see help documentation.
2. The application stops providing the help documentation to the user until requested again.

Use Case #8

1. Name: View Activity Log
2. Participating Actors: Superuser
3. Entry Condition:
 - a. The superuser is logged into the site.
4. Exit Condition:
 - a. The activity log is provided to the superuser.
5. Flow of Events:
 - a. The superuser requests to view the site activity log.

- b. The application shows the site activity log to the superuser.
6. Alternate Flow:

[Cancel]

1. The superuser cancels viewing the site activity log.
2. The application returns the superuser to the previous activity screen.

Use Case #9

1. Name: Restore Previous Version
2. Participating Actors: Superuser
3. Entry Condition:
 - a. The superuser is logged on.
 - b. A previous version of the site exists.
4. Exit Condition:
 - a. The specified previous version of the site is restored.
5. Flow of Events:
 - a. The superuser requests that the application restores the site to a specific version in the activity log.
 - b. The application confirms and restores the site to the previous version in the activity log.
 - c. The application records the restoration in the activity log.
6. Alternate Flow:

[Cancel]

1. The superuser cancels the restoration at intermediate step.

2. The application does not restore and returns the superuser to the previous activity screen.

Use Case #10

1. Name: Register an Administrator
2. Participating Actors: Superuser
3. Entry Condition:
 - a. The superuser is logged in.
 - b. The administrator is not already registered in the application.
4. Exit Condition:
 - a. The administrator is now registered in the application with correct credentials.
5. Flow of Events:
 - a. The superuser requests that another administrator be registered in the application given a set of credentials.
 - b. The application confirms the credentials of the new administrator account and creates the new account.
 - c. The application confirms creation with superuser and records the creation in the activity log.
6. Alternate Flow:

[Cancel]

1. The superuser requests to cancel account creation at intermediate step.
2. The application does not make the account and returns the superuser to previous activity screen.

[User Already Present]

1. The superuser requests to add an account that already exists.
2. The application informs the superuser of duplicate credentials, does not make the account, and returns the superuser to previous activity screen.

Use Case #11

1. Name: Submit an Edit Request
2. Participating Actors: General User
3. Entry Condition:
 - a. The user is viewing the page of the record they think needs an edit.
4. Exit Condition:
 - a. The edit is sent to the superuser for review.
5. Flow of Events:
 - a. The user requests that a record needs to be edited.
 - b. The application allows the user to submit their detailed request to the superuser.
 - c. The application provides the request to the superuser for review.
6. Alternate Flow:

[Cancel]

1. The user cancels the request for edit at an intermediate step
2. The application does not send the request to the superuser and returns the user to the previous activity screen.

Use Case Diagram

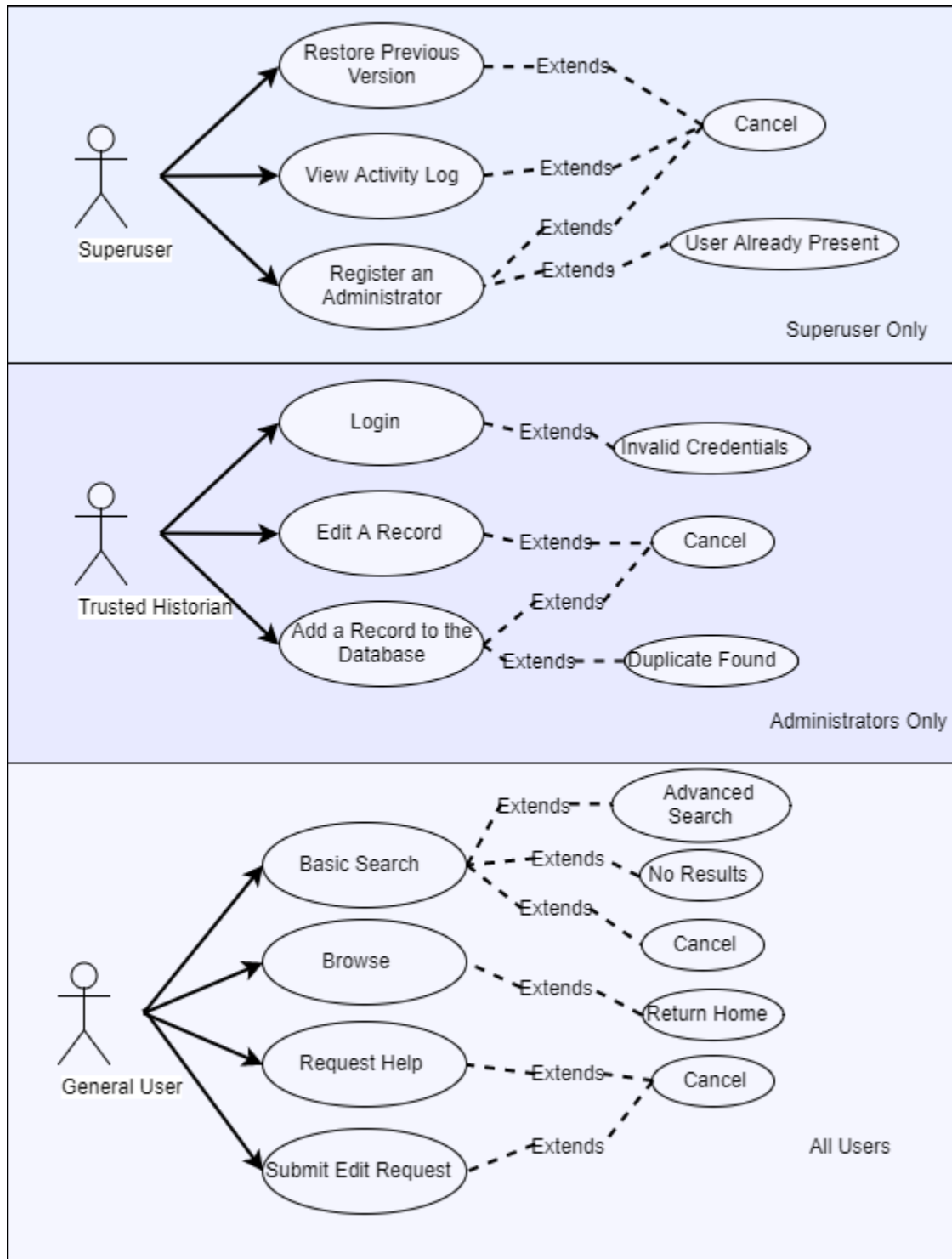


Figure 5: Use Case Diagram

6. Design

6.1 Preliminary Design

For the first iteration, or sprint as it is referred to in SCRUM, we planned to give ourselves three weeks, compared to the one-week length of our other planned sprints, to create a minimal application version as a foundation for the rest of the sprints to build upon. Our goal was to focus primarily on user stories relating to database integrity and setup, simple browsing capabilities, setup viewing capabilities on the website, and establish simple administrator capabilities, such as logging in and editing administrator accounts. We also chose to implement the use cases involving editing a record, registering and logging in as an administrator, and browsing records. These stories and cases represent a minimal version of our application with a functional database and admin capabilities, but only browsing capabilities for the front end. We felt that setting up the back end in the first sprint and leaving the rest of the sprints to really work on the front end was a good plan, because the front end would require the most interaction with our project owner to ensure the site was looking as he intended.

6.2 Initial Architectural Design

To get a feel of our design development throughout the project we compare our first sprint architectural design to our final architectural design. For our initial sprint we created the following entity relationship diagram and class diagram to guide us through the implementation process.

6.2.1 Initial Class Diagram

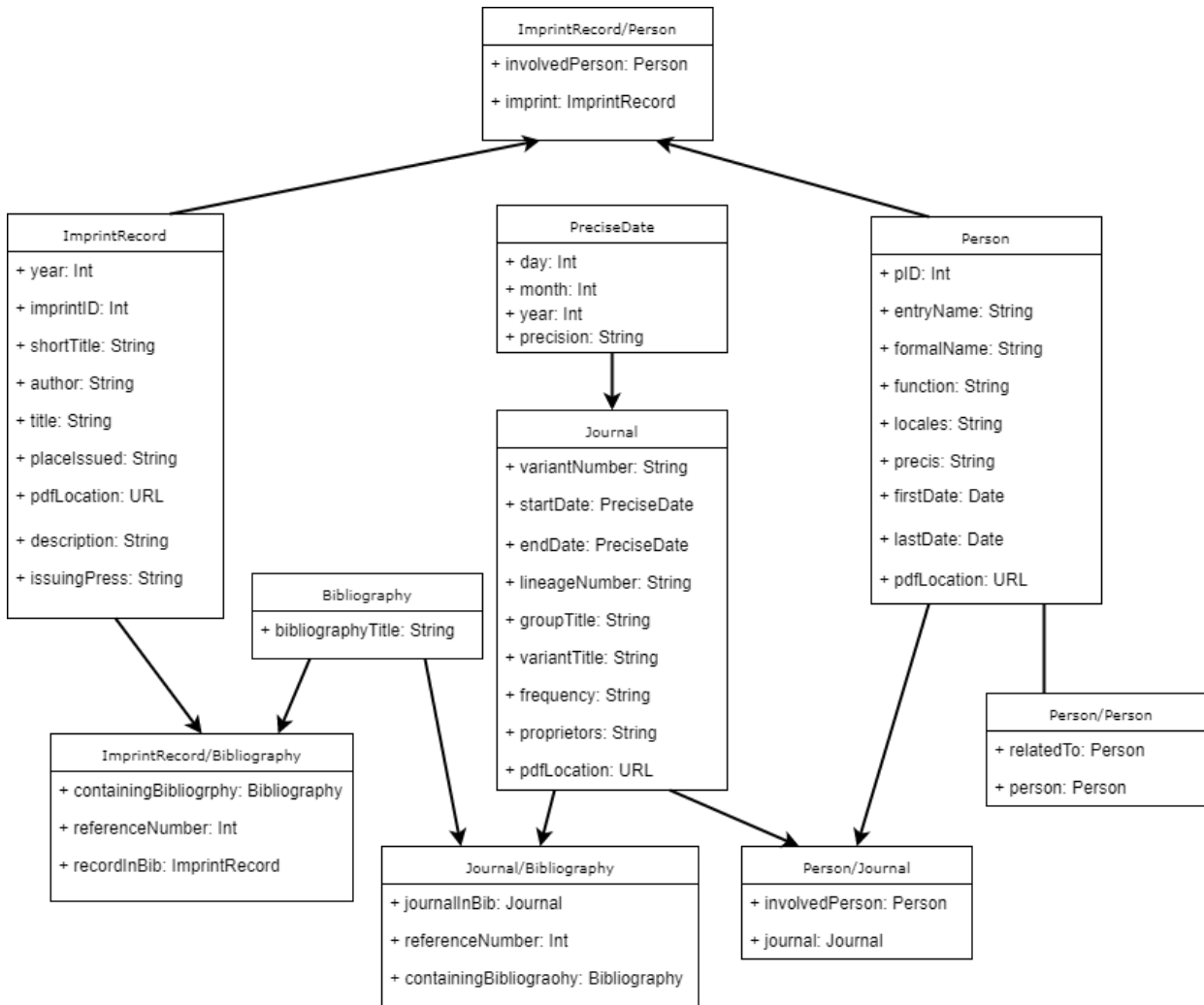


Figure 6: Initial Class Diagram

When we began planning our first sprint we wanted to create a class diagram that would help us develop our database and overall site structure. We began with the three classes Imprint Record, Journal, and Person. All three of these classes would contain all information relating to the physical imprints, newspaper journals, and biographies gathered in the preliminary research done by Dr. Rawson. The fields in these classes represent the actual gathered data regarding a physical document, or person, related to the Virginia printing trade. The classes that we created that includes a “/” in the title, such as Person/Journal, represent the physical relationships

between two of the entities in the research. For example, The Person/Journal class contains a Person and Journal indicating that that Person had some form of relationship with that Journal, possibly as an author, printer, or something of that nature. The purpose of the Precise Date class is to represent some of the vagueness that is inherent to researching with historical dates, as sometimes dates have a level of preciseness, and we felt we should try to represent that in our classes. Finally, the class Bibliography was intended to reference outside bibliographic references that some Journals or Imprint Records contained, such as if that record could be found in the Library of Congress. The separate bibliography class would be used to achieve external cross-referencing through links to the bibliography an imprint or journal was a part of, and we represented this real-life relationship with the ‘ImprintRecord/Bibliography’ and ‘Journal/Bibliography’ classes. We felt it important to pull Bibliography out as there are many different bibliographic reference databases referred to by the research that could later be externally referenced on our site. Overall, we felt this class diagram was a good initial setup for our website as it allowed us to highlight the three main entities of our research while encompassing the many relationships between them. The next step we took in planning our first sprint was to take this initial class structure and apply it to our database in the form of an Entity Relationship Diagram (ERD), which we will show and explain in the next section.

6.2.2 Initial Entity Relationship Diagram

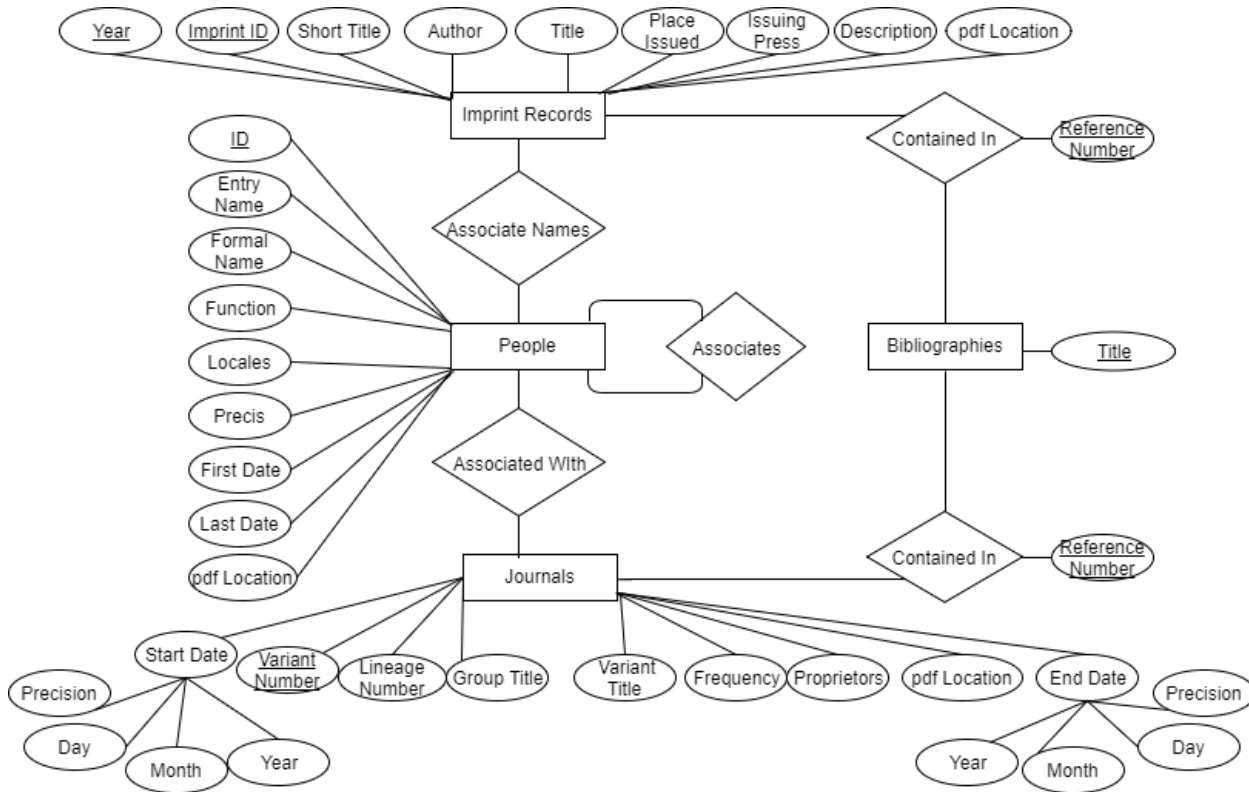


Figure 7: Entity Relationship Diagram

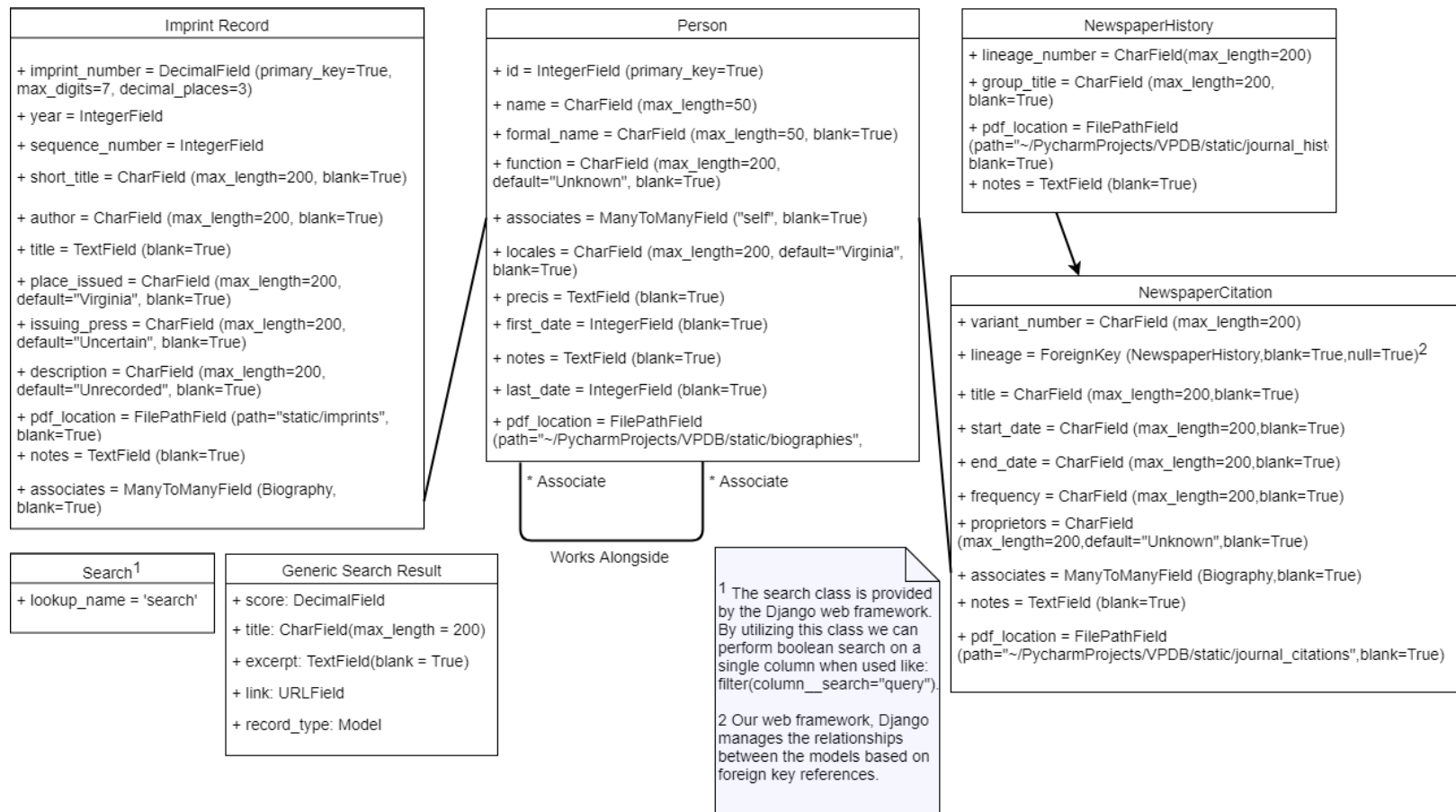
Our initial Entity Relationship Diagram (ERD) does a good job at breaking down our three main models that we described above, Imprint Records, People, and Journals and further shows their relationships that we were trying to capture in our database and classes. Imprint records, Journals, and People refer to the physical records and individuals related to the Virginia printing trade that were researched by Dr. Rawson. Bibliographies refers to the outside sources, or databases, that people can find references to these records. In terms of the relationships, an imprint record can be associated with a person in the print trade, as well as be referenced by a bibliography. A person can have associations with other people in the trade and be associated with an imprint record or journal. A journal can be associated with a person or included in a bibliography. Finally, a bibliography can contain references to both imprint records and journals.

Using both our initial class diagram and entity relationship diagram, we began our Implementation phase, during which we made some design changes that ultimately led us to our finalized design which we will now describe.

6.3 Final Design and Notable Changes

Overall, our final design followed closely to our initial design plan with some minor changes. We completed our first sprint focusing on all the features that we stated in our initial plan that would provide a minimal version. Our ERD did not change at all, but because we did not implement every feature we intended to, our class diagram did change in terms of classes we used and how we represented the relationships between them. We have drawn up our finalized class diagram in the next section to explain in detail the design changes that we made.

Figure 8: Final Class Diagram



6.3.1 Class Diagram

The first obvious change that we made to our class diagram was to remove many of the relationship, or '/', type classes that we had in our initial design. The reasoning behind this is that our web framework we decided on using, Django, does a large amount of relationship management between models for us using foreign key references. The second obvious change was that we ended up not using the Precise Date or Bibliography classes. We did not end up needing those classes as those features were not prioritized for satisfactory completion of the site, and thus were left unimplemented due to time constraints. We also added two new classes to our diagram, Generic Search Result and Search. Generic Search Result was used to make our search feature easier to implement across the various record types. Search was an optional built

in class provided by Django that allowed us to implement Boolean searching. Finally, the last change that we made was to divide our Journal class into two separate classes, Newspaper History and Newspaper Citation. Originally, we had these two classes combined representing the newspaper itself, and lineage the newspaper paper belonged to was simply a field in the class. However, as the project progressed we realized that to accomplish good internal referencing between records we needed to separate the newspaper from the lineage it belonged to. Finally, we added two fields to our three main models, Notes and Associates. The notes field was necessary because we were provided text documents for every record and to capture the full detail about each record, to later be displayed on our site, there was a section of notes that we needed to add into our models. The associates field is how we implemented Django's relationship handling, as it provided the models a way to reference one another without having to have any separate relationship classes. Overall, these are the changes we made to our design throughout our implementation process which we will now break down into greater detail.

7. Implementation

In this section we will explain the steps we took during our implementation phase for the application. We will describe our initial plan for our first sprint, and how that plan was carried out. We also go into detail for each sprint about which user stories were accomplished and provide a burndown chart to conveniently show our projected effort versus what we actual accomplished. After we describe every sprint we will explain how we tested our user stories and evaluate our overall project success.

7.1 Original Expected Project Timeline

For our original expected timeline, we expected to have a first sprint done within 3 weeks of the start of B term. We then expected our subsequent sprints to take about a week and to be close to our final application version three full weeks before the end of B term. Throughout the term we expected to hold weekly meetings as sprint planning meetings, sprint review meetings, and sprint retrospective meetings. We originally planned that for C term we would have a nearly finished application so that we could spend the majority of C term working on this report and our project presentation. Overall, we did not follow the entire planned timing, but we did accomplish our first sprint goals and the overall goal of creating a finalized application by the end of the project. The following are the steps we took towards reaching that goal and the actual timeline that we followed.

7.2 Sprint 0

1. Create index pages
2. Upload .pdfs of every record to Reclaim Hosting
3. Define URL patterns for the site
4. Create record placeholder pages
5. Create test objects with some .pdfs
6. Configure app to change names to beta domain
7. Create .csv files for data to be added to database

The goal of our initial sprint was to set up a bare minimum application so that we could expand upon it weekly in the sprints that followed it. To prepare for the first sprint, we had a small iteration before our first that was simply to set up the structure of our site architecture. It did not cover any specific user stories, but it was critical to starting our application. We listed the objectives that we met during this initial sprint above. Our first obstacle that we encountered was adding the original research files to the database from an excel sheet so that they could be accessed by the site. The MySQL database that we used was able to handle .csv files so we wanted to convert our provided spreadsheets into this format. We had a few problems matching column values to class attributes and had to pull some of the data out into separate columns so that we could use it in our classes. For example, we had to take the name field in the Biography sheet we were given and use that to pull out First Name and Last Name fields. For this reason, we did not finish adding our records to our database in this sprint but did create the .csv files that would allow them to be added later. We then used Django to create index pages, defined our URL patterns, and created placeholder pages. Finally, we wanted to have access to our data on Reclaim itself so we added our pdf files to the site to be referenced later and created some test objects using Django's ORM (to see the code implementation for this see 'populate_db.py' in our submitted code repository), so we could test our site's general linking and referencing between two model objects. We have also included a screenshot below of these test objects, showing how

they were initially displayed on our site. By the end of this sprint we had a version of our site that was very minimal and could display our model relationships correctly with the test objects. Looking to our burndown chart below, our initial sprint was a success in terms of effort as we basically met our expected effort throughout the entire sprint. Leaving this sprint, our major goal was to add the data into the models using the .csv files as that would give us a completed initial version.

Index of Virginia Printing

This page lists all the records in the database.

Newspaper Citations

[Link to the Index of Newspaper Citations](#)

- [Abingdon](#)

Newspaper Histories

[Link to the Index of Newspaper Histories](#)

-

Biographies

[Link to the Index of Biographies](#)

- [Adams](#)

Imprint Records

[Link to the Index of Imprint Records](#)

- [Example Imprint Record](#)

Figure 9: The minimal Index page displaying links to the four test objects (Sprint 0)

Iteration 0 Burndown

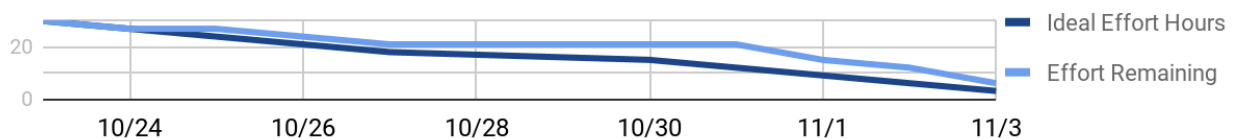


Figure 10: Sprint 0 Burndown Chart

7.3 Sprint 1

1. As a superuser, I want the application to allow me to save data records on a remote server in order to ensure data longevity.
2. As a general user, I want the application to save a permanent URL for a record so that I can return to where I left off in research or share my findings with others.
3. As a superuser, I want to be able to add a new superuser in order to allow the site application to change ownership.

4. As a superuser, I want to be able to remove a superuser in order to allow the site application to change ownership.
5. As a superuser, I want to be able to update a superuser in order to allow the administrator details to stay accurate.
6. As an administrator, I want to remove a record, regardless of type, in order to keep the collection accurate.
7. As a general user, I want to browse by type of record in order to quickly find relevant records.

Sprint 1 is where a bulk of our site was created, and the bare minimum foundation was laid for the rest of the project. In this sprint we allowed records to be saved to the database and used the .csv files to add all the initial records to the database. We have included a screenshot below to show how the records on the site were initially displaying their fields on their own respective pages. The records that we added also could be removed in this sprint. For our administrator user stories, we completed the tasks of adding, editing, and removing administrators by using Django's built in admin pages. Finally, we also accomplished basic browsing by being able to view all the records in our database in an index page. Originally, we had intended to accomplish more in terms of record manipulation this sprint, however it was more complex than we thought, and for that reason our burndown chart shows we had to accomplish a lot of our tasks mid-way through the week instead of steadily accomplishing them throughout the entire week. With the ending of this sprint providing us records to manipulate and a page structure, the sprint that followed was focused on formatting the browsing abilities and administrative record manipulation.

Adams

Name: Adams

First Date: 1700; **Last Date:** 1830

Function: Unknown

Locales: Virginia

Adams is associated with 0 other people.

Adams is associated with 0 newspaper variants.

Adams is associated with 0 imprint records:

[Link to a full biography.](#)

[Go back to Index of Biographies](#)

Figure 11: Detailed biography page for Adams, showing all basic data (Sprint 1)

Iteration 1 Burndown

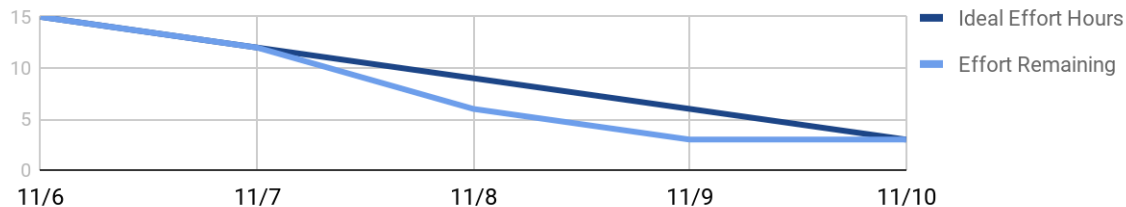


Figure 12: Sprint 1 Burndown Chart

7.4 Sprint 2

1. As an administrator, I want to add an imprint to the database application in order to expand the applications knowledge base and keep it up to date.
2. As an administrator, I want to add a biography to the database application in order to expand the applications knowledge base and keep it up to date.
3. As an administrator, I want to add a newspaper lineage to the database application in order to expand the applications knowledge base and keep it up to date.
4. As an administrator, I want to edit existing imprints in order to keep the collection accurate and up to date.
5. As an administrator, I want to edit existing newspaper lineages in order to keep the application accurate and up to date.
6. As an administrator, I want to edit existing biographies in order to keep the application accurate and up to date.
7. As a general user, I want to browse records in chronological order in order to gather an understanding of the data and what it represents.

This sprint we chose to work on the admin site side of the application in order to implement record manipulation, such as adding or editing records in the database using the site interface. On the admin site, we successfully added the ability to add and edit all record types in

the database with the proper credentials, and the format in which record editing is displayed to the user is shown below in Figure 13. We also worked on the main site format by adding a chronological browsing page that let us view imprint records by date, mimicking a timeline, which can be seen in Figure 14 below. Our site was coming along pretty much to plan at this point, we had a lack of accomplished effort, or tasks, in the latter half of this sprint because record manipulation was quite difficult to implement, but we started the week out strong by tackling the browsing first. Our third sprint planning meeting is when we decided we needed to add a notes field to every record to capture information not displayed in the other fields, and we also decided that we needed to implement the cross-referencing between records in our third sprint.

The screenshot shows the Django administration interface for editing a Biography. The page title is "Django administration" and the user is logged in as "ADMIN". The breadcrumb trail is "Home > Virginia printing > Biographies > Henry Tutwiler". The page title is "Change biography". There are two buttons: "HISTORY" and "VIEW ON SITE". The form fields are:

- Id:** 554
- Name:** Henry Tutwiler
- Formal name:**
- Function:** Publisher
- Locales:** Harrisonburg
- Precis:** Publisher of a Presbyterian sermon in Harrisonburg in 1815.

Figure 13: The admin interface for editing a Biography (Sprint 2)

Imprint Records by Year Published

- 1730**
- [Typographia: An ode, on printing.](#)
 - [Session Laws \(May 1730\).](#)
 - [New Virginia Tobacco-Law.](#)
 - [Charge to the Grand Jury.](#)
- 1731**
- [Virginia and Maryland Almanack \[1732\]](#)
 - [Virginia Miscellany](#)
- 1732**
- [The Complete Mariner](#)
 - [Dialogue between Thomas Sweet-Scented...](#)
 - [Dialogue between Thomas Sweet-Scented...](#)
 - [Dialogue between Thomas Sweet-Scented...](#)
 - [Acts of Assembly, 1732](#)
 - [Session Laws \(May 1732\)](#)
 - [Speech of the Governor to the General Assembly \(May 1732\)](#)
 - [Reply to the Governor by the Burgesses in Assembly \(May 1732\)](#)
 - [Journal to the House of Burgesses \(May 1732\)](#)
- 1733**
- [Short and Easy Method with the Deists](#)
 - [Revisal of the Laws, 1733](#)
- 1734**
- [Every Man His Own Doctor](#)
 - [Every Man His Own Doctor](#)
 - [Session Laws \(August 1734\)](#)
 - [Speech of the Governor to the General Assembly \(August 1734\)](#)
 - [Reply to the Governor by the Council in Assembly \(August 1734\)](#)

Figure 14: Chronology page displaying imprint records (Sprint 2)

Iteration 2 Burndown

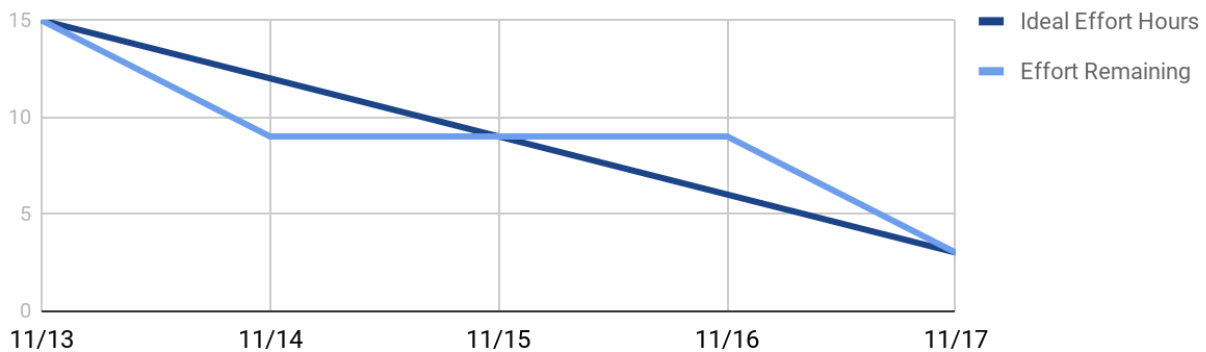


Figure 15: Sprint 2 Burndown Chart

7.5 Sprint 3

1. As a general user, I would like to be able to view the related narrative history, individuals, and imprint records of a newspaper lineage.
2. As a general user, I would like to view related imprint records, individuals, and newspaper variants to a specific individual.
3. As a general user, I would like to view related individuals to a specific imprint record.

With the goal of cross-referencing in mind we started our third sprint purely focused on completing the models of every record on the site and connecting them to one another. This took us a long time as we had to receive feedback from our client on how records should be associated and how the field for each records note should be added to the record pages. We then had to take our clients feedback and edit our site structure based upon it. Our biggest challenge was how to convey all the details of a record in a way that was sensible and visually appealing for the users, and we did not complete this task until our fourth sprint, explaining our bumpy task completion throughout the week and why it ended on us not having the expected amount of tasks, or effort, completed that we would have liked. During this sprint we did accomplish the setup for displaying record details however, as we wrote python scripts to translate the provided spreadsheets into data that could be added to MySQL. By adding the data to our MySQL database, we would be able to format the data in the upcoming sprint to display all the details about each model. Additionally, we had to add the relationships between entities, such as associations between two tradespeople, to accomplish cross-referencing between two records on our site. The format we chose for cross-referencing was to include links on each record page that could bring a user to the related record pages, how we chose to display this can be seen in Figure 16 below. Overall, this sprint was a success because it set up our plans for our next sprint and accomplished a major application requirement of cross-referencing.

Seymour P. Charlton is associated with 3 other people.

- [Augustine Davis](#)
- [Hamilton Shields](#)
- [William C. Shields](#)

Figure 16: A portion of the detailed biography page showing page links (Sprint 3)

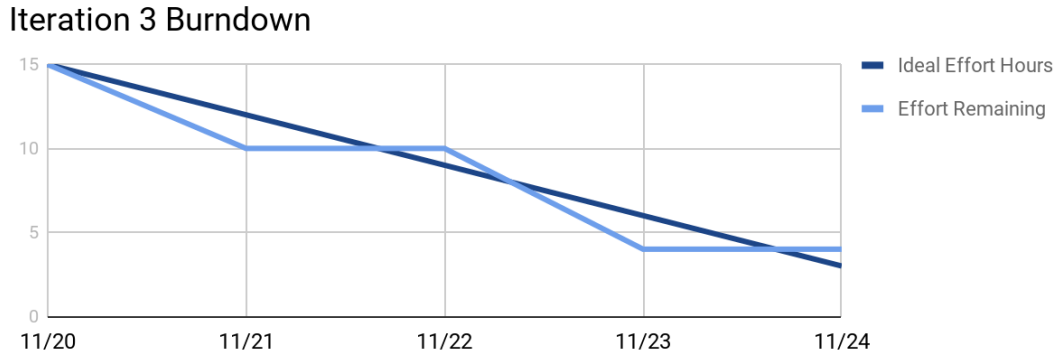


Figure 17: Sprint 3 Burndown Chart

7.6 Sprint 4

1. As a general user, I would like to view notes for newspaper citations.
2. As a general user, I would like to view notes for newspaper lineages.
3. As a general user, I would like to view notes on imprint records.
4. As a general user, I would like to view notes and personal data for people in the database.

We began sprint 4 by finishing our record page design in a way that pleased our client and conveyed the data of each record broken up by model fields. An example of this can be seen in Figure 18 below. We chose to display the record pages with each section being a field of the model because we felt it was the easiest way to convey the most information to a user about a record without having it be an overwhelming amount of data. Our next focus was making sure every field from our models was imported into our site, including the “Notes” field found in our models. This task was challenging because we had to find a way to pull a block of text from a converted .pdf files and keep some logical structure. This task took us a long time to plan as we had to find a way to make every record’s “Notes” be similar in structure. Our solution was to write a python script for each record type that could pull the notes from a plain text file, which we got by converting every .pdf file, into an excel sheet to be converted to a .csv file that we could then import into MySQL. We then merged the notes data with the data already on the site

by manually manipulating the database. Overall, this one task took us the entire forth sprint to complete and thus our effort burndown is very under the goal until the end of the week when we finished our planned tasks. Upon completion of this sprint we had our completed models on the site and set our goals to our final tasks of search functionality and user interface design.

Adams

Name: Adams

First Date: 1796; **Last Date:** 1796

Function: Publisher

Locales: Staunton

Precis

Partner of printer & publisher John Wise (455) in Staunton's Virginia Gazette in late 1796.

Notes

Publisher Staunton Partner of printer & publisher John Wise (455) in Staunton's Virginia Gazette in late 1796. The brevity of the Wise & Adams partnership - less than six months - suggests that Adams was editor and/or financier of Wise's paper once Staunton merchant Robert Douthat (147) withdrew from the venture earlier that year. His absence from other Staunton imprints makes further identification difficult, especially given the common nature of his surname. It is possible that he was the William Adams (an Irishman from Belfast), a visitor who attended meetings of Staunton's Masonic Lodge in early 1797, where Wise and Douthat were leading presences; if so, then he was also likely a transient figure in the town. No Personal Data discovered. Sources: Imprints; Brigham; Brown, Freemasonry in Staunton.

Adams is associated with 1 other people.

- [Johann Weiss](#)

Adams is associated with 1 newspaper variants.

- [The Staunton Gazette](#)

Adams is associated with 0 imprint records:

[Printer-Friendly PDF](#)

[Go back to Index of Biographies](#)

Figure 18: Individual biography page for Adams, displaying all data (Sprint 4)

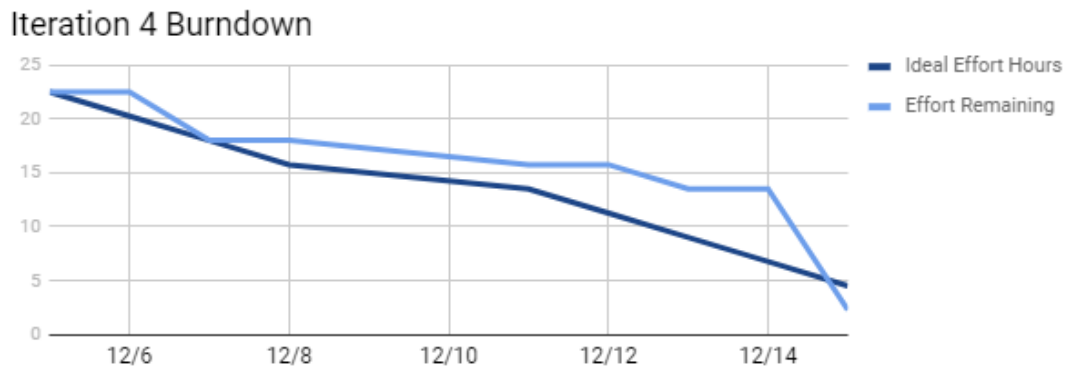


Figure 19: Sprint 4 Burndown Chart

7.7 Sprint 5

1. As a general user, I would like to search for a term in newspaper citations.

2. As a general user, I would like to search for a term in newspaper lineages.
 3. As a general user, I would like to search for a term in imprint records.
 4. As a general user, I would like to search for a term in an individual's record in the database.
 5. As a general user, I would like to have accessibility features on each page, such as a toolbar.
 6. As a general user, I would like to search specific fields in newspaper citations for a term.
 7. As a general user, I would like to search specific fields in newspaper lineages for a term.
 8. As a general user, I would like to search specific fields in imprint records for a term.
 9. As a general user, I would like to search specific fields in an individual's record for a term.
-

Search was the hardest part of our implementation process as it provided the most technical difficulty. For this sprint we took the 6 weeks of C term to implement full-text search, field-based search, as well as create usability features for our site that included a toolbar. We divided search into the two separate categories of full text, which means searching all the models and all the fields for a string, and field-based search, meaning only searching certain fields for a provided phrase. Our full text search did not take too long to implement and is simply a search bar on the bottom of our sites homepage that takes an inputted search term and searches every model's title, author, and notes fields for that value.

While full text search did not take too long to implement, it did take a very long time to figure out the logic behind field-based search because there are many possible ways to implement such a feature. The result of how we chose to format our field-based search can be seen in Figure 21 below. We opted to have every field be a check box on the left side of the page, broken up by model type. A user could then select the fields they wanted to either search or not search and enter a new keyword in a search bar at the bottom of the left pane to begin a new refined search.

We also had to consider formatting of our site and how a user would view the page and search results. Overall, this was the most satisfying feature to finish and brought our application from the bare minimum to the final version, the completed look of our site homepage can be seen in Figure 20 below. We used a stylesheet and the same header on each page to implement the navigation toolbar and design on every page of our site, giving it a uniform layout throughout. The navigation toolbar contains links to the homepage, the various browsing index pages, and the about page for the project, it can be seen in both figures below. The search in the header is another full text search that brings the user to the search results page, where they can conduct a refined search, just like the search bar on the main homepage.

This sprint took much longer than expected to implement and had three big tasks in it that each took a large amount of time to work on and were each finished at different times, which is reflected in our burndown chart below. We had periods of time where we would only work on one of the tasks and this severely limited the total amount of tasks we completed throughout the sprint. If we had to redo this project we would have probably planned to give these tasks more time, or break them into smaller tasks, as they were very critical to the completion of our project and we felt that with better planning they could have been implemented quicker and with more detail.

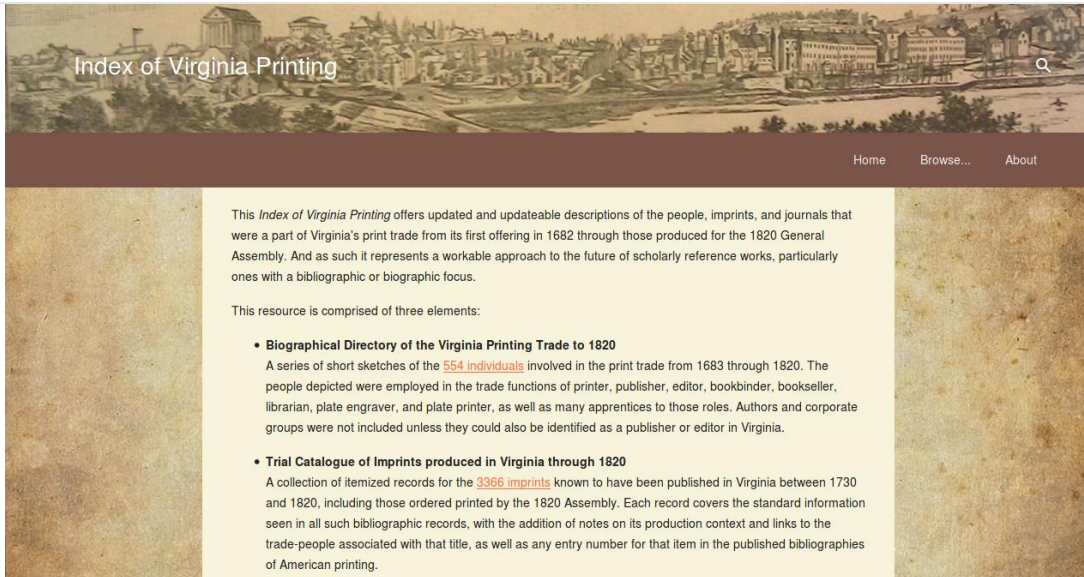


Figure 20: The index page after the 5th Sprint

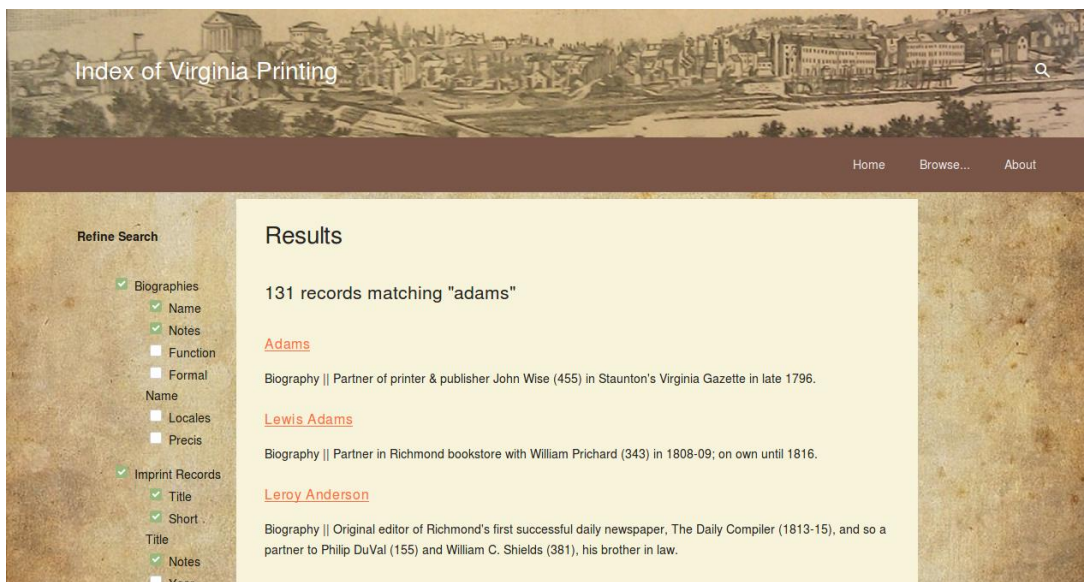


Figure 21: The results displayed, next to "Refine Search" form (5th Sprint)

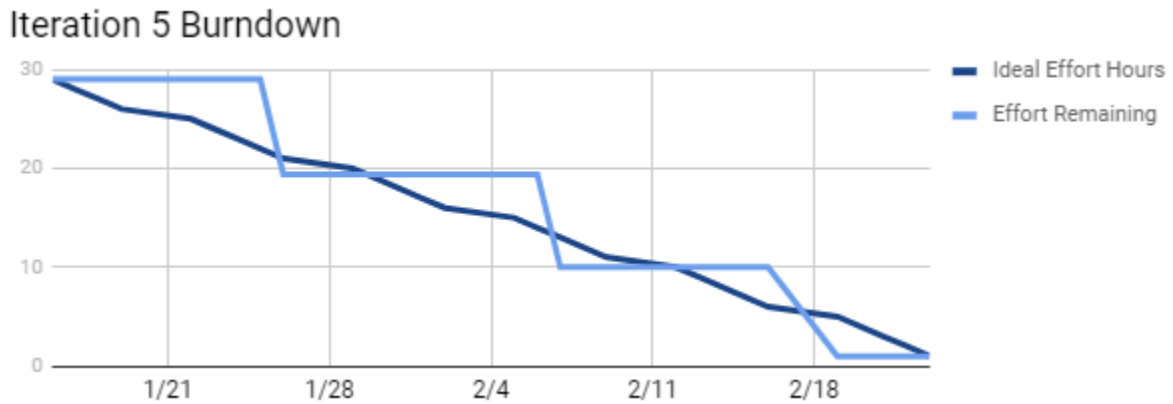


Figure 22: Sprint 5 Burndown Chart

8. Testing

To test the correctness of our application we used informal validation and user acceptance tests. The primary things we checked were that individual record pages displayed the correct information from the spreadsheets provided by Dr. Rawson, that links between pages worked correctly, that the admin could make changes that were visible to a user on the site, and that search returned appropriate results. When extracting data from the spreadsheets and text files we verified correct output on at least 1-2 records of each type before attempting to extract from all records. Once the data was on the website we viewed at least 3 records of each type randomly and verified the information was displayed as we expected. We know that these tests are not comprehensive but were unable to complete more in depth testing due to project time constraints.

To verify that the admin could make visible changes to the content of the website we logged into the administrator site and added test records through the admin interface. Then we viewed these records on the site, confirming the data we added displayed. We would then delete the test record to test the admin's ability to delete records, all these user level tests were successful.

To validate search, we tested simple queries on a subset of the data. For most of search testing we only looked at biographies. For example, we searched for names known to be in the database, such as “Adams”, and verified the application returned all instances of biographies with the name “Adams”. We then expanded these tests by changing which record types and fields we searched. Overall, each search provided the correct results and we could confirm that our search functionality was behaving correctly/

Our testing proved that the application behaved correctly in displaying, searching, and extending the records in the database. Our client also performed acceptance tests and usability tests from the perspective of a historian and confirmed the site was working as intended.

9. Assessment

Overall, we met our minimal functional and nonfunctional requirements and the project was an overall success. We successfully created a site that was accepted by our project owner that contained all the minimal functional requirements, including storing data, retrieving data, entering data, authorizing accounts, site administration, and site cross-reference ability. We did not get to fully complete site administration, as there are some features on the administration site which were not implemented, such as creating a trusted historian. We also did not fully accomplish cross-reference ability since the site only cross-references itself, and not outside bibliographic references. With more time we would have loved to implement these functionalities.

In terms of non-functional requirements, longevity and scalability were met by our choice in application architecture. Our choice of Django allowed us to make certain that our application would be updatable for at least 10 years, the initial goal, as there are built in ways for this

application to allow for administrative users to update the site continuously. Our security requirement ended up not being a priority, but we feel we met the minimum requirement as general users cannot edit our site records and people cannot gain access to edit records without proper credentials. Usability was also confirmed in user acceptance testing when our client confirmed the project was working as intended and considered to be an overall success.

We were successful in our non-functional project goals, but we did have some originally intended functionalities that we had to leave out. Such features included providing different types of administrators, viewing all records on a map, and citation links to outside references. If we could do the project over we would probably try to start our search feature much sooner, as this is what took up a majority of our project time. We also would try to limit our goals early on to keep focused on the primary functionalities of the site.

10. Future Work

We hope to see this project handed to another group of students in the form of an IQP or MQP. We feel that there are some unfinished user stories that really would help the site stand out, such as viewing records on a map, that we just did not have time to complete. We are very happy with where our work has ended up, but we feel there are many opportunities for expansion. Our user manuals are tools that we hope can guide potential future designers in furthering this application's capabilities. An IQP would be great for this as there is a lot of background in place already and they would have to just research a few of the tools to start expanding on our site's foundations.

11. Conclusion

By the completion of this project our team successfully accomplished our goal of creating a historical database backed website, containing the records relating to the Virginia printing trade prior to 1820, for our client Dr. Rawson that can be searched, extended, and is user friendly. By researching the records provided by Dr. Rawson and the layouts and features of similar historical sites, we were able to develop a vision for our site's final appearance and function that would guide our implementation process. We also researched what frameworks, servers, and databases would best suit this type of project, and ultimately, we decided a Django application framework hosted on Reclaim Hosting with a MySQL backend would be the best fit for our requirements. We then chose to utilize a SCRUM methodology throughout our design and implementation phases to track our requirements and ensure that none of the essential ones were left out. Our primary focus was to create our site with features that allowed it to be searched, extended, and easily used and we accomplished this in five sprints spanning the course of B and C term. Although we did not achieve every proposed feature, we did achieve our core functionality goals by creating a site that was searchable, expandable, and user friendly. Overall, our client and team both consider the project, and final application, a success. It is now our hope that our application design process can serve as a template for others to preserve various types of historical records using a similar method as we have used in our project. Our team has gained a new insight into the creation and design of websites, especially those backed by databases, and we hope to use this information in the future to help create and teach others how to create sites like the one created with this project.

12. References

Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. "Agile Manifesto." accessed October 15.

Chronicling America. 2017. "Chronicling America." Library of Congress, accessed October 9. Rails. 2017. "Welcome to Rails." Ruby on Rails, accessed October 15.

Clinger, Will. 2016. "Disadvantages of the Scrum Process." Northeastern University accessed 15 October.

Clinger, Will. 2016. "Advantages of the Scrum Process." Northeastern University accessed 15 October.

Encyclopedia Virginia. 2017. "Encyclopedia Virginia." Virginia Foundation for the Humanities, accessed October 9.

Eriksson, Ulf. 2015. "The Difference Between Functional and Non-Functional Requirements." ReQtest, accessed October 23.

Hughey, Douglas A. 2009. "The Traditional Waterfall Approach." University of Missouri-St. Louis, accessed October 14.

Hughey, Douglas A. 2009. "Scrum." University of Missouri-St. Louis, accessed 15 October.

IT Knowledge portal. 2017. "Software Development Methodologies." Association of Modern Technologies Professionals, accessed October 9.

Price, Gene Hill. 1999. "The waterfall model." Old Dominion University, accessed 14 October.

Prosoft Nearshore. 2015. "5 Types of Scrum Meetings." Prosoft, LLC., accessed 15 October.

Rawson, David Andrew. 2016. *An Index of Virginia Printing*. Virginia: The College of William & Mary.

Rouse, Margaret. 2017. "database (DB)." SearchSQLServer, accessed October 9.

Stellman, Andrew. 2009. "Requirements 101: User Stories vs. Use Cases." WORDPRESS, accessed October 9.

Techopedia. 2017. "Software Framework." Techopedia Inc., accessed October 9.

Virginia Historical Society. 2017. "Exhibitions." Virginia Historical Society, accessed October 9.

Appendix 1

Biography (People) Data Fields:

ID: Unique identifier assigned to each individual trade-person.

Entry Name: The most common form of name of the individual associated with printing trade¹ in the form (last name, first name).

Formal Name: Full name of the individual, if amends/differs from entry name, in the form (first name, last name, honorific).

Function: Relationship of the individual to print trade (i.e. printer, publisher, editor, bookbinder, bookseller, librarian, plate engraver, and plate printer, or apprentices to those roles).

Locales: Places where individual practiced the print trade (i.e. Lewis Adams published a newspaper in the locale of Richmond).

Precis: A short statement that explains the relation of individual to all their associated fields in plain text.

First Date: Earliest known date individual was practicing print trade.

Last Date: Last known date individual practiced print trade.

IVP Associates: Other individuals listed in the database that this individual shared a relationship to in the printing trade.

Newspaper(Journal) Data Fields:

Lineage No.: Unique identifier for each family of newspaper titles, assigned in chronological order by locale and earliest date initial journal appeared there, covering all subsequent changes

¹ 'printing trade' throughout this appendix refers to the Virginia printing trade pre-1820's.

in title and/or ownership. For example (Lynchburg 04).

Group Title: Short title assigned to this family of newspapers, usually the most familiar title form during journal's lifetime.

Variation No.: Chronological numbering of titles seen in this newspaper family, corresponding to changes in journal's title, publication frequency, and ownership.

Variation Title: Title employed by this variation number.

Start Date: Earliest date that this variation title was published.

End Date: Latest date that this variation title was published.

Frequency: How often this variation title was published (weekly, twice weekly, thrice weekly, daily, etc.).

Proprietor(s): Name of publisher as reported in this variation title.

IVP Associates: Individuals involved in production of this variation title, using corresponding identifier number in Biography table.

LC Control: Control number (aka LCCN) assigned to this variation title by Library of Congress in its MARC-format electronic-catalogue-record².

Brigham: Reference number assigned by Clarence S. Brigham in his *History and Bibliography of American Newspapers, 1690-1820* (1962).

Cappon: Reference number assigned by Lester J. Cappon in his *Virginia Newspapers, 1821-1935: A Bibliography with Historical Introduction and Notes* (1936).

Norona-Shelter: Reference number assigned by Delf Norona and Charles Shetler in their *West Virginia Imprints 1790-1863: A Checklist of Books, Newspapers, Periodicals, and Broadsides* (1958).

² MARC format is the standard form employed in modern electronic imprint records.

Sabin: Reference number assigned by Joseph Sabin in his *Bibliotheca Americana: A dictionary of books relating to America, from its discovery to the present time* (1868-1936).

Evans: Reference number assigned by Charles Evans in his *American Bibliography: A Chronological Dictionary of All Books, Pamphlets, and Periodical Publications Printed in the United States of America from the Genesis of Printing in 1639 down to and including the Year 1820* (1903-1959).

Imprint Data Fields:

Imprint No.: Unique two-part identification number assigned to this imprint, beginning with the year that this title was issued, followed by number representing the approximate place in the chronology of publications produced during that year (i.e. 1730.001 or 1731.001).

Short Title: A shortened title assigned to this imprint, designed to facilitate searches and standardize minor variations in titles in a recurring imprint series, such as government documents.

Author: The author (or authors) of this imprint reported in this record.

Title: Full transcription of the long-form as printed on the title-page of this imprint.

Place Issued: The city or town in Virginia where this imprint was issued.

Issuing Press: Printer or press office issuing this imprint.

Description: Reports physical characteristics of this imprint, including page count, page size, and binding.

IVP Associated Names: Individuals involved in production of this imprint, using corresponding identifier number in Biography.

EAI Series #: Indicates whether this imprint was filmed as part of the Early American Imprint series (a subscription service offered by Readex/Newsbank), and reports reference number that was assigned to that item, also indicates if filmed. Initially just two series, based on numbering seen in the Evans bibliography; a third series followed (**Readex/EAI Supplements**) employing new numbers not seen in Evans.

Evans: Reference number assigned by Charles Evans in his *American Bibliography: A Chronological Dictionary of All Books, Pamphlets, and Periodical Publications Printed in the United States of America from the Genesis of Printing in 1639 down to and including the Year 1820* (1903-1959).

Bristol: Reference number assigned by Roger P. Bristol in his *Supplement to Charles Evans' American Bibliography* (1970).

Shipton-Mooney: Usually the reference number assigned by Charles Evans in his *American Biography*, as summarized by Clifford K. Shipton and James E. Mooney in their *National Index of American Imprints Through 1800: The Short-title Evans* (1969).

Shaw-Shoemaker: Reference number assigned by Ralph R. Shaw and Richard H. Shoemaker in their *American Bibliography: A Preliminary Checklist for 1801-1819* (1958-1966).

Shoemaker: Reference number assigned by Richard H. Shoemaker in his *A Checklist of American imprints for 1820-1829* (1964-1973).

Readex/EAI Supplements: [see EAI above].

Norona-Shetler: Reference number assigned by Delf Norona and Charles Shetler in their *West Virginia Imprints 1790-1863: A Checklist of Books, Newspapers, Periodicals, and Broad-sides* (1958).

Clayton-Torrence: Reference number assigned by William Clayton-Torrence in his *Trial Bibliography of Colonial Virginia* (1908).

Swem, Virginia: Reference number assigned by Earle Gregg Swem in his “A Bibliography of Virginia,” in three parts (1910-1919).

Swem, Conventions: Reference number assigned by Earle Gregg Swem in his “A Bibliography of the Conventions and Constitutions of Virginia” (1910).

Drake, Almanacs: Reference number assigned by Milton Drake in his *Almanacs of the United States* (1962).

Hummel, Southeastern: Reference number assigned by Ray O. Hummel, Jr. in his *Southeastern Broadsides Before 1877: A Bibliography* (1971).

Hummel, Virginia: Reference number assigned by Ray O. Hummel, Jr. in his *More Virginia Broadsides Before 1877* (1975).

Rink, Tech Americana: Reference number assigned by Evald Rink in his *Technical Americana: a checklist of technical publications printed before 1831* (1981).

Wegelin, Am Poetry: Reference number assigned by Oscar Wegelin in his *Early American Poetry: a compilation of the titles of volumes of verse and broadsides ... issued during the seventeenth and eighteenth centuries* (1903).

Sabin, Bib Americana: Reference number assigned by Joseph Sabin in his *Bibliotheca Americana: A dictionary of books relating to America, from its discovery to the present time* (1868-1936) for the national bibliography list compiled by Joseph Sabin.

Lincoln, Am Cookbooks: Reference number assigned by Waldo Lincoln and Eleanor Lowenstein in their *American Cookery Books, 1742-1860* (1954) for the national cookery book list compiled by Waldo Lincoln and Eleanor Lowenstein.

Adams, Am Pamphlets: Reference number assigned by Thomas Randolph Adams in his *American Independence: The growth of an idea; a bibliographical study of the American political pamphlets printed between 1764 and 1776 dealing with the dispute between Great Britain and her colonies* (1965).

Winans, Book Catalogs: Reference number assigned by Robert B. Winans in his *Descriptive Checklist of Book Catalogues Printed Separately in America, 1693-1800* (1981).

Wright, American Fiction: Reference number assigned by Lyle Henry Wright for his *American Fiction, 1774-1850: A Contribution toward a Bibliography* (1969).

Ford, Ratification Biblio.: Reference number assigned by Paul Leicester Ford in his *Bibliography and Reference List of the History and Literature relating to the Adoption of the Constitution of the United States 1787-8* (1896).

Eng Short Title Catalogue: Reference number assigned by the English Short Title Catalogue project, managed by the British Library and the Research Libraries Network of the United States. ESTC is a record of titles of imprints published between 1473 and 1800 in Britain and North America, and primarily in English. Data is available through the website of the British Library.

Walgren, Freemasonry: Reference number assigned by Kent Logan Walgren in his *A Bibliography of pre-1851 American Scottish Rite Imprints (non-Louisiana)* (1994).

Wheeler, MD Press: Reference number assigned by Joseph Towne Wheeler in his *The Maryland Press, 1777-1790* (1938).

Lowens, Songsters: Reference number assigned by Irving Lowens in his *The American Songster before 1821: A List of Incomplete and Unlocated Titles* (1960).

McDonald, Carriers' Addresses: Reference number assigned by Gerald D. McDonald, Stuart C. Sherman, and Mary T. Russo in their *A Checklist of American Newspaper Carriers' Addresses, 1720-1820* (2000).

Ford, Biblio. Hamiltoniana: Reference number assigned by Paul Leicester Ford in his *Bibliotheca Hamiltoniana: A List of Books Written by or relating to Alexander Hamilton* (1886).

Austin, Medical Imprints: Reference number assigned by Robert B. Austin in his *Early American Medical Imprints: A guide to works printed in the United States, 1668-1820* (1961).

Skeel, Mason Locke Weems: Reference number assigned by Emily Ellsworth Ford Skeel in her *Mason Locke Weems, His Works and Ways* (1929). [PLF was Skeel's brother; when he died, she completed his unfinished bibliography]

Welch, Amer Children's Books: Reference number assigned by A. S. W. Rosenbach in his *Early American Children's Books* (1933).

Winans, Book Catalogues: Reference number assigned by Robert B. Winans in his *A Descriptive Checklist of Book Catalogues Printed Separately in America, 1693-1800* (1981).

Bötte & Tannhof, German Printing: Reference number assigned by Gerd-J. Bötte and Werner Tannhof in their bibliography appended to Karl John Richard Arndt and Reimer C. Eck, *The First Century of German Language Printing in the United States of America* (1989).

Albaugh, Religious Periodicals: Reference number assigned by Gaylord P. Albaugh in his *History and Annotated Bibliography of American Religious Periodicals and Newspapers, established from 1730 through 1830* (1994).

Singerman, Judaica Americana: Reference number assigned by Robert Singerman in his *Judaica Americana: A bibliography of publications to 1900* (1990).

Amer. Broadsides & Ephemera: Latest Readex/Newsbank project, accessed through the Archive of Americana, as subscription service. Includes: “American Broadsides and Ephemera, Series I; Civil War Broadsides and Ephemera; The American Civil War Collection, 1860-1922: From the American Antiquarian Society; Afro-Americana Imprints from the Library Company of Philadelphia, 1535-1922.”

Appendix 2

Table 1: Excerpt from Imprint Dataset A-G

Imprint No.:	Short Title:	Author:	Title:	Place Issued:	Issuing Press:	Description:	IVP Associated Names:
1730.001	Typographia: An ode, on printing.	Markland, J. [John Markland, attorney, New Kent]	Typographia: An ode, on printing: Inscib'd to the Honourable William Gooch, Esq; His Majesty's lieutenant-governor, and commander in chief of the colony of Virginia.	Williamsburg	William Parks	16 pgs.; 23 cm. (4to).	William Parks (321)
1730.002	Session Laws (May 1730).	Virginia. General Assembly.	All the publick acts, made at a session of the assembly, begun and held at the city of Williamsburg, in Virginia, on Thursday the twenty-first day of May, 1730.	Williamsburg	William Parks	Unknown	William Parks (321)
1730.003	New Virginia Tobacco-Law.	Virginia. General Assembly.	The new Virginia tobacco-law, made at the last session of assembly.	Williamsburg	William Parks	Unknown	William Parks (321)
1730.004	Charge to the Grand Jury.	Gooch, William, Sir (1681-1751), lieutenant governor.	Charge to the Grand Jury: At a General Court held at the capitol of the city of Williamsburg, in Virginia, on Monday the 19th of October, 1730. By the Honourable William Gooch, Esq; governour of Virginia. : Publish'd at the request of several gentlemen.	Williamsburg	William Parks	4 pgs.	William Parks (321)
1731.001	Virginia and Maryland Almanack [1732]	Warner, John	The Virginia and Maryland Almanack. Shewing the time of sun rising and setting, length of days, new and full moon, eclipses, fixt and moveable feasts, seven stars rising and setting, weather, days of the several courts, &c. For the year of our Lord Christ, 1732	Williamsburg	William Parks	22+ pgs. [evidently 36 pgs. total]	William Parks (321)

Table 2: Excerpt from Imprint Dataset H-Q

EAI Series #	Evans:	Bristol:	Shipton-Mooney:	Shaw-Shoemaker	Shoemaker	Readex/EAI Supplements	Norona-Shetler:	Clayton-Torrence	Swern, Virginia:
1 3298	3298							114	
								117	22509
								115	22510
1 3370	3370							113	
1 40011		B892	40011						

Appendix 3

Table 3: Excerpt from Journals Dataset A-G

Start Date	End Date	Frequency	Proprietors	IVP Associates	LC Control	Brigham
January 4, 1806	ca. August 1812	Weekly	John G. Ustick	John Gano Ustick (421); Thomas Ustick (528)	90069188	2 1105
September 7, 1811	late 1819	Weekly	John G. Ustick	John Gano Ustick (421); Thomas Ustick (528)	90069191	2 1105
February 5, 1784	July 4, 1789	Weekly	George Richards & Co.	George Richards (355), Thomas Bond (039)	84024726	2 1112
July 30, 1789	ca. November 1793	Weekly	Hanson & Bond	Samuel Hanson (200) Thomas Bond (039)	84024725	2 1111

Appendix 4

Table 4: Excerpt from People Dataset A-F

ID	Entry Name	Formal Name	Function	Locales	Precis
001	Adams		Publisher	Staunton	Partner of printer & publisher John Wise (455) in Staunton's <i>Virginia Gazette</i> in late 1796.
002	Adams, Lewis		Bookseller, Publisher	Richmond	Partner in Richmond bookstore with William Prichard (343) in 1808-09; on own until 1816.
003	Adrian, Robert		Bookseller, Bookbinder	Alexandria	Employee in Alexandria bookstore of Peter Cottom (107) & John A. Stewart (402) in 1806.

Appendix 5

Historian User Manual

This user manual accompanies the Virginia Printing Database (VPDB) application and will help visitors to the website and trusted historians navigate the application.

Part 1: Website Visitor

The first category of users is a visitor to the website who wants to learn about the history of the printing trade in pre-1820 Virginia. This could be a student, researcher, genealogist, or just someone who is curious. This user has experience in using the Internet and search engines but does not have a technical background.

The first page of the website (Figure 23, <https://indexvirginiaprinting.org/>) gives some background on the project and explains the types of records available in the database. From there, users may browse and search through these records.

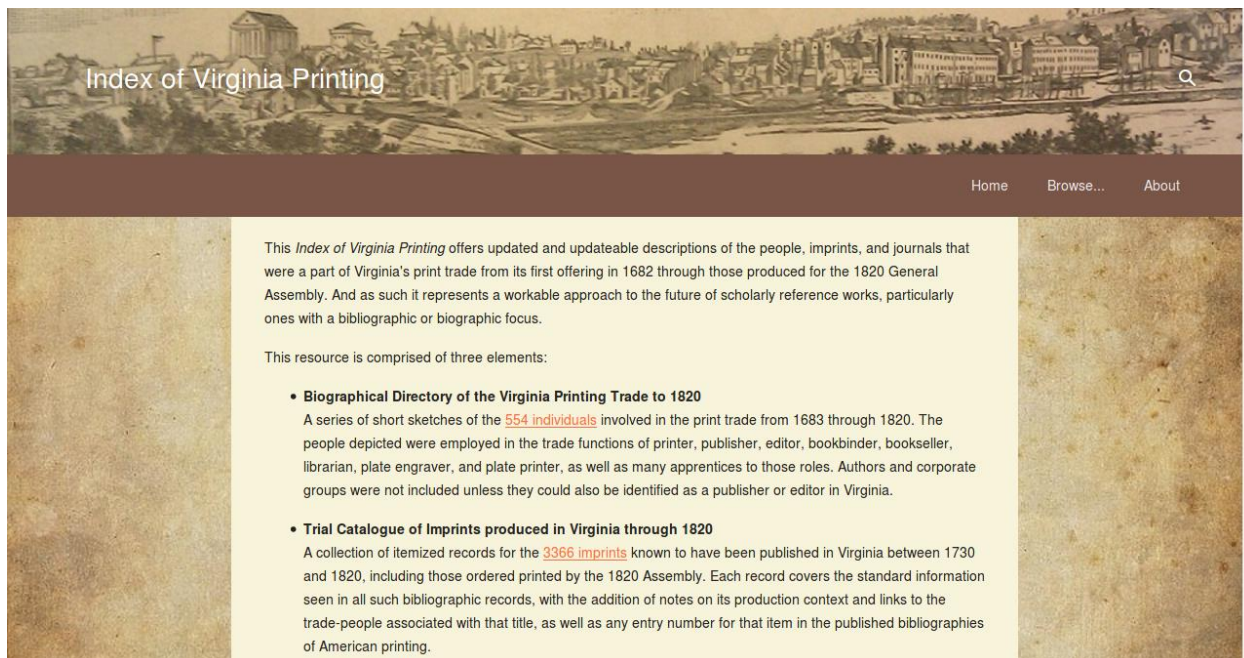


Figure 23: The homepage of the Index of Virginia Printing

Browsing Records

Browsing and viewing records on the website is very easy. There are many ways to browse the website, such as by record type, chronologically, or following related links.

Viewing Records

Each of the four record types has an index listing all records of those types. You can access these pages under the “Browse...” button in the navigation toolbar. These four pages are:

- **Biographical Directory:** <https://indexvirginiaprinting.org/bios/>
- **Catalogue of Imprints:** <https://indexvirginiaprinting.org/imprints/>
- **Histories of Newspaper Lineages:** https://indexvirginiaprinting.org/news_hists/
- **Citation of each Newspaper Variant:** https://indexvirginiaprinting.org/news_cites/

The directory pages are split into pages to manage the huge number of records in the database, so you will need to click through “next” page to see them all. Through these links you will find data about each record in the database. For example, by clicking the first link in the catalogue of imprints, you will see the data associated with “Typographia: An ode, on printing” (Figure 24, <http://indexvirginiaprinting.org/imprint/1730.001/>).

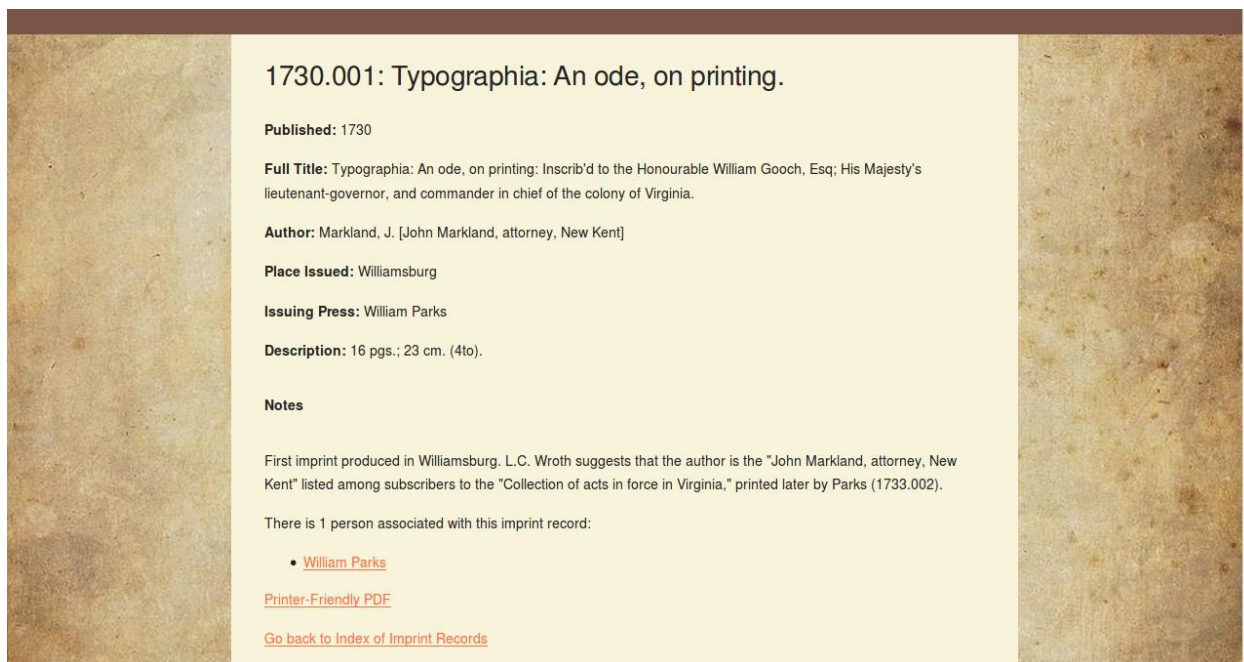


Figure 24: Imprint “Typographia: An ode, on printing” available on the *Index*

Each record also provides a link to a Printer-Friendly PDF which contains all the information on the record's page but formatted for printing in black and white.

Chronology

The Imprint records are also available sorted by publication date (Figure 25,

<http://indexvirginiaprinting.org/chronology/imprints/>). You can access this page through the

“Browse...” button in the navigation toolbar.

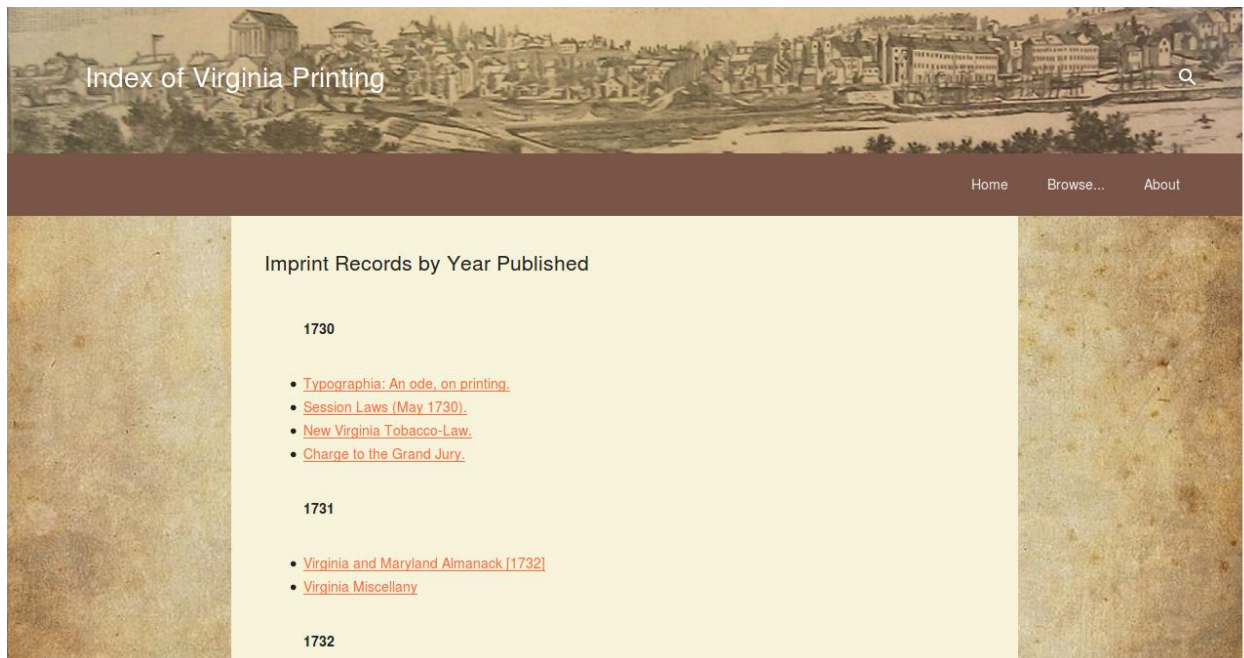


Figure 25: Browse imprint records by the year they were published.

Linking associations

Almost all records in the database are linked with others. At the bottom of each record page is a list of the biographies, imprints, or newspapers the record is associated with. All of these are hyperlinks to the associated record. For example, the page for William Parks (Figure 26, <http://indexvirginiaprinting.org/bio/321/>) has links to the 3 tradespeople, 1 newspaper variant, and 121 imprint records he is associated with. This provides a way to navigate through related records, and to cross-reference related information.

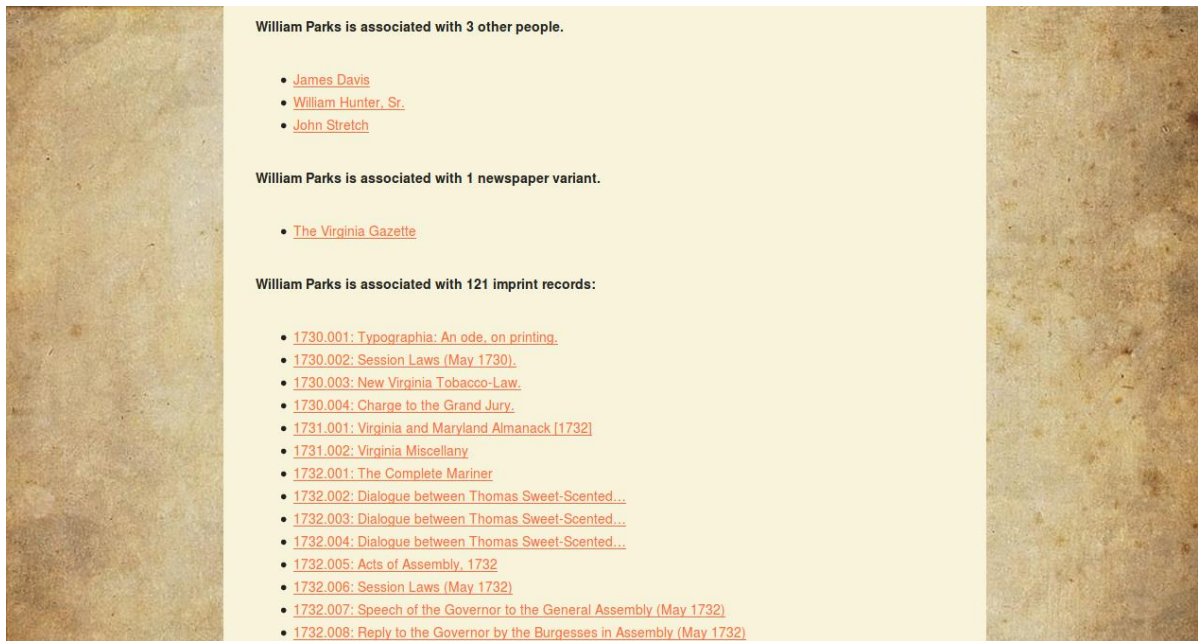


Figure 26: Excerpt from the page for William Parks showing some of his associations

Searching Records

If you are interested in searching for a subset of the records available, the *Index* provides a search function on all the records in the database. This search is simple but will allow you to ensure the results are related to your interests. All search is case-insensitive, so a search for “john” and “John” are the same search.

There are two ways to initiate a search. The first is from the index page in the search box at the bottom of the page (Figure 27). The second is the magnifying glass in the top right corner of the header (Figure 28). When clicked, it will expand into a search box where you can type your query. This ensures that you are always able to initiate a search from anywhere on the web application.

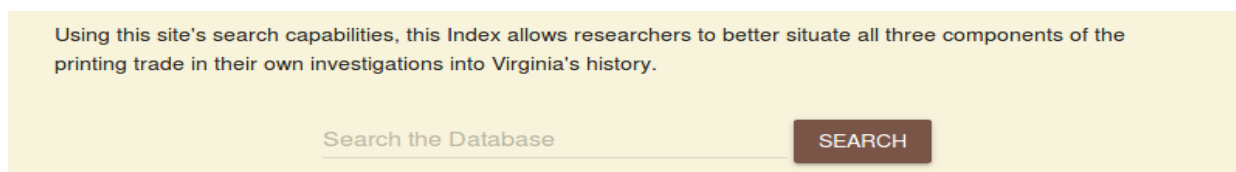


Figure 27: The search box available on the Index Page



Figure 28: The location of the magnifying glass in the webpage header.

Default Search

When you initiate a search from the header or index page, the application searches the database for instances of your query. The search does not search all possible properties of the records. By default, it will search the name or title of a record and the 'notes' section. Specifically, it will search the following for any word in the query:

- Biography
 - Name
 - Notes
- Imprint Record
 - Full Title
 - Short Title
 - Notes
- Newspaper Citations
 - Title
 - Notes
- Newspaper Histories
 - Group Title
 - Notes

It will separate the query into words and search for *any* of them, so “John Quincy Adams” becomes “john” OR “Quincy” OR “Adams”. If records of more than one type match the search query, they will all be in the results. You may need to go through multiple pages to find the results you are interested in.

All searches are encoded in the URL, so search results may be shared. For example, if you copy the URL of the search results for “Adams” (Figure 29,

http://indexvirginiaprinting.org/search/?search_term=adams) you may share it with someone else or save it for later.

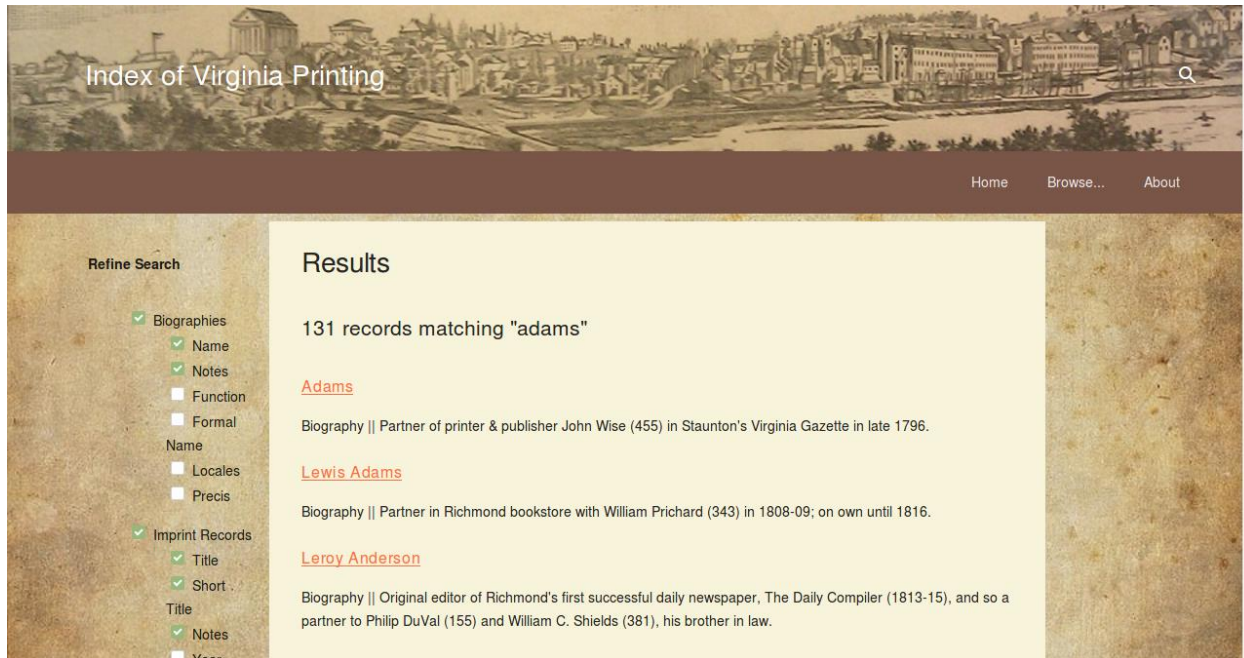


Figure 29: The default search results for “Adams”

Limiting Records & Fields Searched

After a basic search, the results page includes a section that allows you to refine your search (left side of Figure 29, above). This series of checkboxes allows you to limit or expand your searches. If you uncheck any of the 4 top-level record types, the application will not search within those record types. You may also check or uncheck properties of the records, such as biography locales, to include or exclude them in the search. For example, if you are interested in the biographies of editors, you can check 'Biographies' and 'Function' and search for 'editor' (Figure 30).

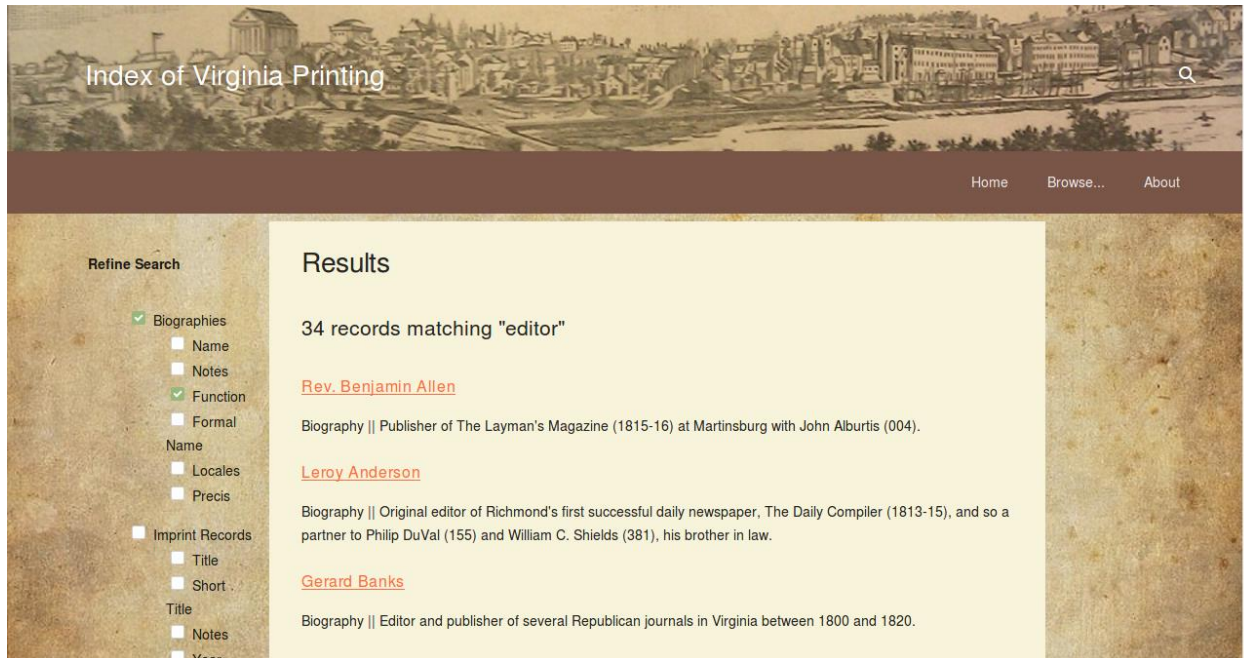


Figure 30: Limiting the search for biographies who have a function listed as “editor”

Boolean Search

The application also supports a simple implementation of Boolean search. This allows you to be more specific about records that should be returned. You can use Boolean search in the following ways:

- **Search for a phrase:** if you include quote marks (“”) around a phrase, the entire phrase will be treated as one search word. For example, searching for “John Quincy Adams” (with quotes) will only return records with the entire name of the 6th president, whereas searching without quotes will give results for records that include *any* of the names “John” “Quincy” or “Adams”.
- **Require a word be in the search results:** If you include a plus sign (+) before a word, all results must include it. For example, if you search for '+John +Quincy +Adams', all three names must be in the record, but they may not necessarily be in that order.

- **Eliminate a word from the search results:** If you include a minus sign (-) before a word, the record will not include it. However, this is limited to a single property of the record. For example, if you search 'Adams -Lewis' on the biography name field only, the result is the record for 'Adams'. If you include the biography notes field in the search, 'Lewis Adams' will also be in the results because the search is true on the notes field (the notes for Lewis Adams never include the word 'Lewis').

Combined with field search, Boolean search allows highly specific searches.

Part 2: Trusted Historian

A trusted historian is someone who has administrator access to the website. They can create, update, and remove records from the database.

Logging In

The administrator website is available at <https://indexvirginiaprinting.org/admin>. First, you must log into the administrator website (Figure 31). Enter the username and password for the administrator site. There are no account recovery options if you do not know the credentials. Once logged in, you will be able to see the site administration page (Figure 32).

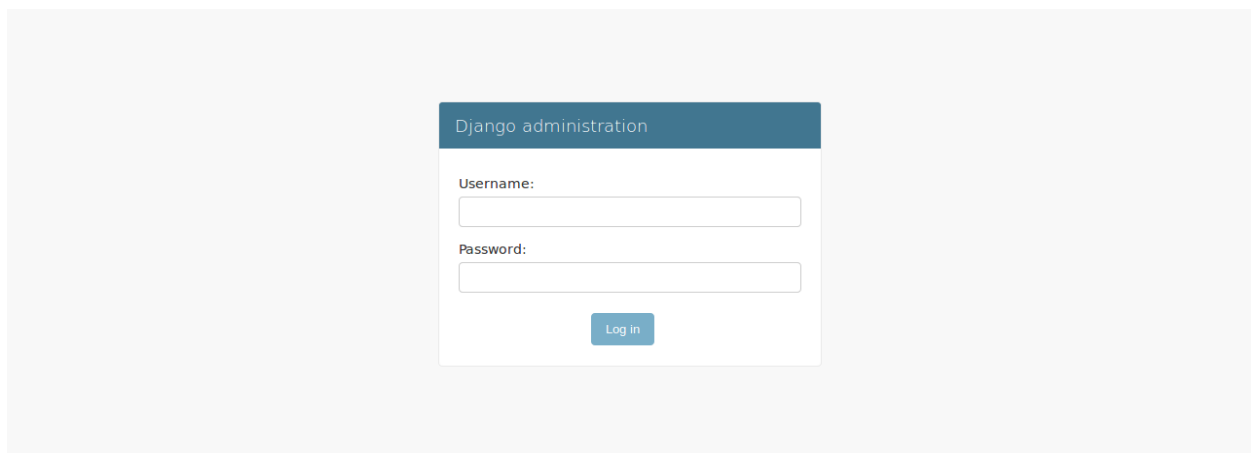


Figure 31: The login page for the administrator website

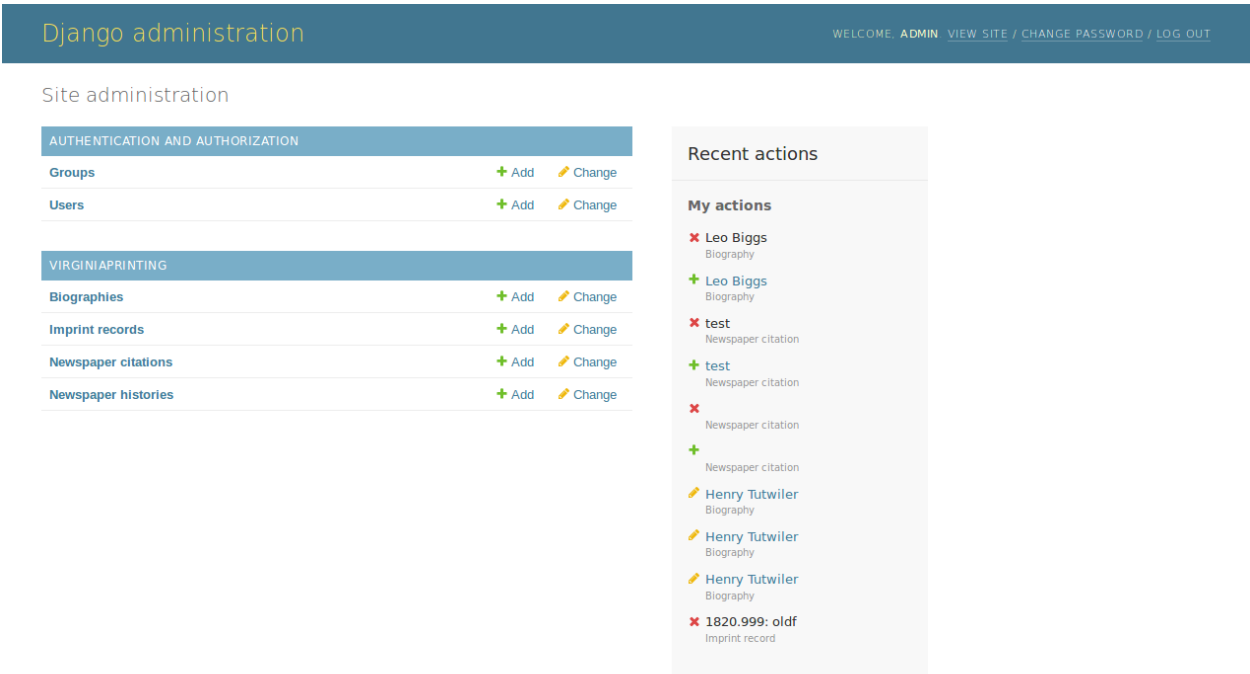


Figure 32: The landing page for the administrator website.

Updating Records

The trusted historian can use the administrator page to change the records in the *Index* without having to directly manipulate the database. They can create new records, edit existing records, and delete records.

Create

To create a new record, click the “Add” button in the row corresponding to the type of record you want to add (Figure 33). You will be taken to a new page which has a form to fill out (Figure 34). The **bold** fields are required. For the biography record, you can add many different associates by holding CTRL and clicking names in the “associates” list. You may also add an associate on the fly by clicking the green plus sign next to the list of existing associates. When you are ready to add the record to the database, click “Save” on the bottom right. If you provide

an invalid value, such as an ID that already exists in the database, you will get an error when you try to save the record (Figure 35).

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add Change
Users	+ Add Change
VIRGINIA PRINTING	
Biographies	+ Add Change
Imprint records	+ Add Change
Newspaper citations	+ Add Change
Newspaper histories	+ Add Change

Figure 33: Add a Biography by clicking the “Add” button in that row

Django administration WELCOME, ADMIN [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > [Virginia](#) > [Biographies](#) > Add biography

Add biography

Id:

Name:

Formal name:

Function:

Locales:

Precis:

First date:

Last date:

Associates:

Adams
 Lewis Adams
 Robert Adrian
 John Alburitis
 William Baron Allegre
 Rev. Benjamin Allen
 William Allen
 David Ammen
 John Ammen
 John Anderson

+
Hold down "Control", or "Command" on a Mac, to select more than one.

Pdf location:

Notes:

Figure 34: A blank biography record allows the trusted historian to add new records.

Django administration WELCOME, ADMIN [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > [Virginia](#) > [Biographies](#) > Add biography

Add biography

Please correct the error below.

Biography with this Id already exists.

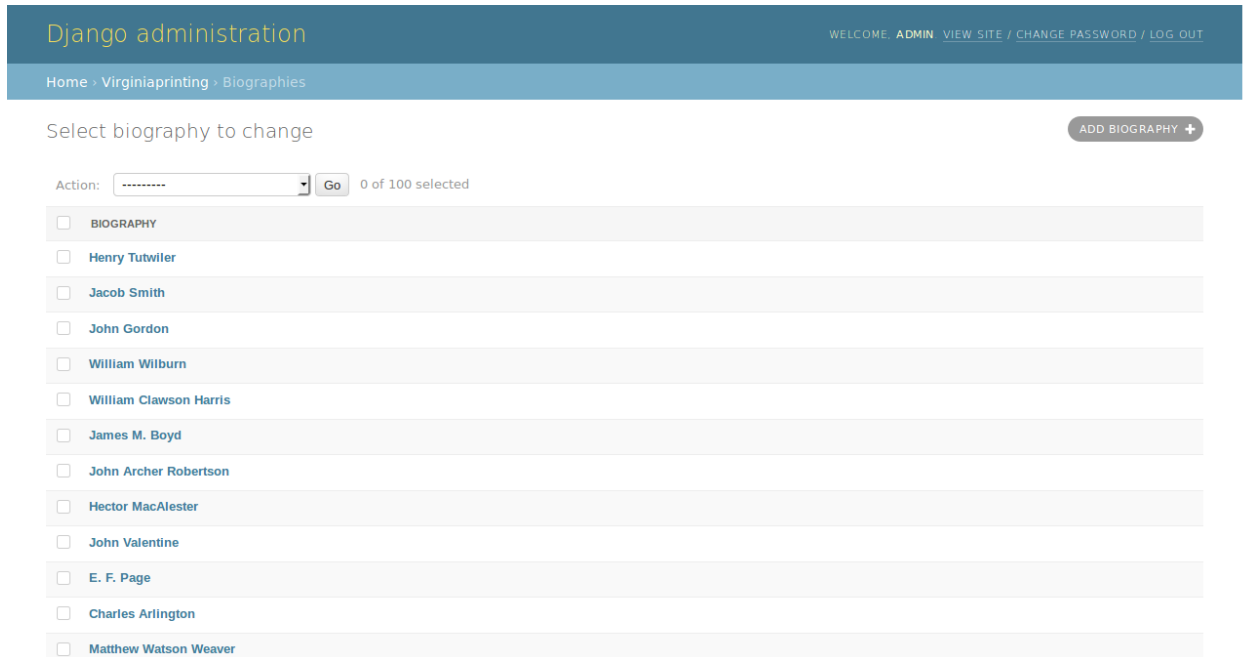
Id:

Name:

Figure 35: An error occurred trying to save this record because the ID is already in use

Edit

The trusted historian may edit records by clicking the “Change” button in the appropriate row. The application will then show you a list of existing records (Figure 36). When you click on a record to edit it, the form will auto-populate with the existing values for that record (Figure 37). When you have made your changes, click “Save”.



The screenshot shows the Django administration interface for a biographies database. At the top, there is a header with "Django administration" on the left and "WELCOME, ADMIN VIEW SITE / CHANGE PASSWORD / LOG OUT" on the right. Below the header, a breadcrumb trail reads "Home > Virginiaprinting > Biographies". The main content area is titled "Select biography to change" and includes an "ADD BIOGRAPHY +" button. Below this, there is an "Action:" dropdown menu set to "-----" and a "Go" button, with the text "0 of 100 selected" to the right. A list of biographies follows, each with a checkbox and a name: BIOGRAPHY, Henry Tutwiler, Jacob Smith, John Gordon, William Wilburn, William Clawson Harris, James M. Boyd, John Archer Robertson, Hector MacAlester, John Valentine, E. F. Page, Charles Arlington, and Matthew Watson Weaver.

Figure 36: The list of biographies in the database. Select one to edit.

Change biography

[HISTORY](#) [VIEW ON SITE](#) >

Id:	<input type="text" value="554"/>
Name:	<input type="text" value="Henry Tutwiler"/>
Formal name:	<input type="text"/>
Function:	<input type="text" value="Publisher"/>
Locales:	<input type="text" value="Harrisonburg"/>
Precis:	<input type="text" value="Publisher of a Presbyterian sermon in Harrisonburg in 1815."/>
First date:	<input type="text" value="1815"/>
Last date:	<input type="text" value="1815"/>
Associates:	<div><ul style="list-style-type: none">Franklin VioletAnderson M. WaddillJoseph Cunningham WaggonerJames WalkerSamuel WalkupWilliam WalkupGiles Ward, Jr.Benjamin WarnerJohn WarrockLaurentz Wartmann</div>
Pdf location:	<input type="text" value="554 Tutwiler Henry.pdf"/>
Notes:	<div><p>Publisher Harrisonburg Publisher of a Presbyterian sermon in Harrisonburg in 1815. Henry Tutwiler was not a print tradesman, but rather the patron for a work issued in 1815 from the Harrisonburg press of Lawrence Wartmann (431).</p><p>The sermon Tutwiler published then – Gratitude to God, urged from a Review of the Late War; and the Restoration of National Peace. A discourse delivered in Harrisonburg; April 13, 1815 – had been given by Rev. George Bourne (043). Along with Rev. Andrew B. Davidson (116), Bourne had previously owned Wartmann's press; these Presbyterian evangelicals had induced the printer to relocate to Harrisonburg in early 1813 to conduct their Theological Printing Office, an office intended to produce</p></div>

[Delete](#)[Save and add another](#)[Save and continue editing](#)[SAVE](#)

Figure 37: The form populates with the record's data and may be changed graphically.

Delete

A trusted historian may delete a record from the page used to make changes to it (Figure 37, above) by clicking the red “Delete” button in the bottom left corner. They may also delete multiple records at once. In the listing of the records, check the box corresponding to the records to be deleted (Figure 38) and select “Delete records” from the “Action” dropdown menu.

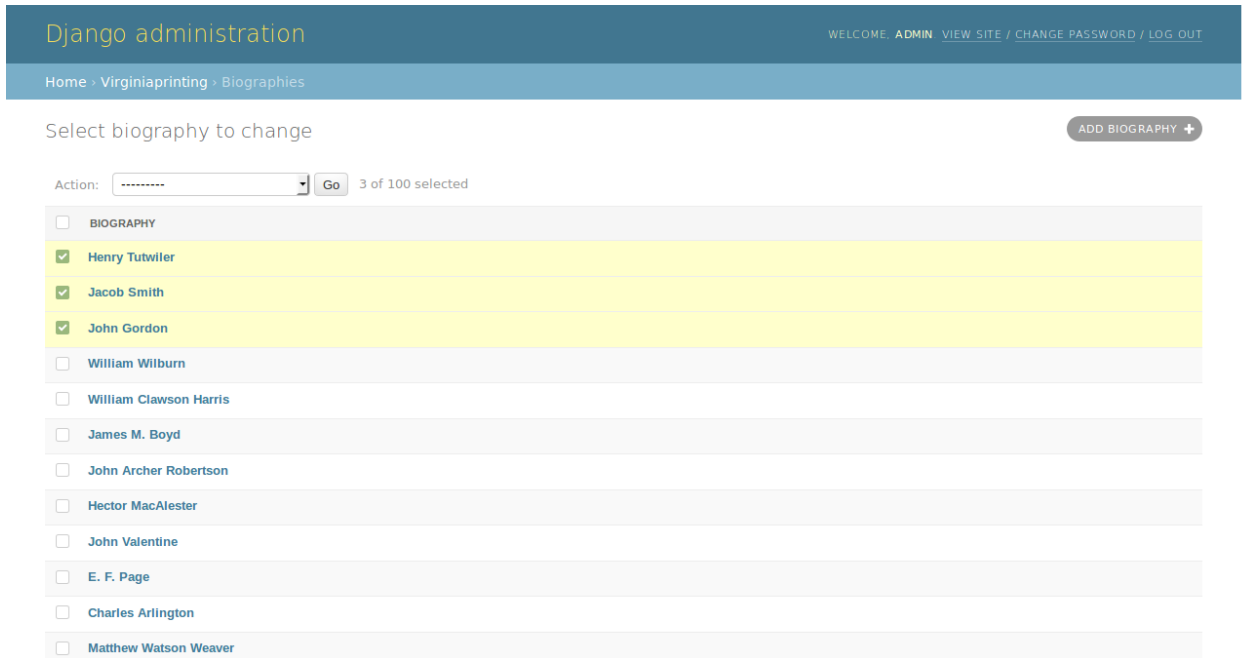


Figure 38: Select multiple records to delete and choose “Delete” in the dropdown menu

Administrator History

The application will save a record of the changes made to the database, so if unauthorized changes were made through the administrator interface they will appear under “Recent Actions” (Figure 39). However, these do *not* act as “restore” points for the records: if an item has been deleted it cannot be added back and clicking a record that has been changed will only show the current content of that record.

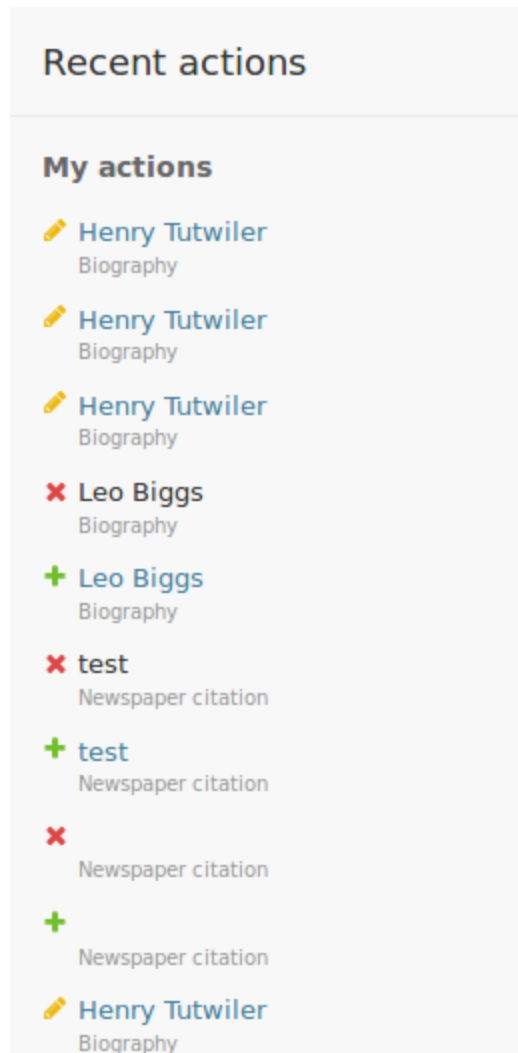


Figure 39: The “Recent Actions” list shows records have been added, edited, and deleted.

Appendix 6

Administrator User Manual

This user manual accompanies the Virginia Printing Database (VPDB) application and will help the administrator learn how to set up and manage the application.

Assumed Background

To manage the application, the admin does not require any specific technical experience.

However, when installing the application, the administrator is assumed to have some background in web servers and access to a remote server running cPanel³. The hosting service should also have a MySQL⁴ database available. In the initial production version, the hosting provider was Reclaim Hosting⁵. The following walkthrough will explain how VPDB was deployed to Reclaim Hosting. A similar process may be followed for other remote servers running cPanel.

The administrator will additionally need access to data, in the form of CSV files, to populate the application. As the historical data is the intellectual property of the project's client, it is not provided with the documentation or application code.

To summarize, deploying the application will require:

- A server running cPanel
- An instance of MySQL
- Data to populate the website with
- MySQL Workbench⁶, for directly manipulating the database

³ <https://cpanel.com/>

⁴ <https://www.mysql.com/>

⁵ <https://reclaimhosting.com/>

⁶ <https://www.mysql.com/products/workbench/>

Installing the Application

This section will explain how to deploy VPDB onto a new server running cPanel.

Getting the Code

There are many options for getting the code onto the server. Two ways are explained below. The first is to download the code locally and upload to cPanel. Alternatively, if the administrator is comfortable using the command line, they may log into the server and download the code directly onto the server.

Via cPanel

1. Download the Code

The first step is to download the application code. This is hosted on GitHub at <https://github.com/lbowenbiggs/virginia-printing-db>. Click the green “Clone or download” button and select “Download ZIP” (see Figure 40). Save the file to your local computer.

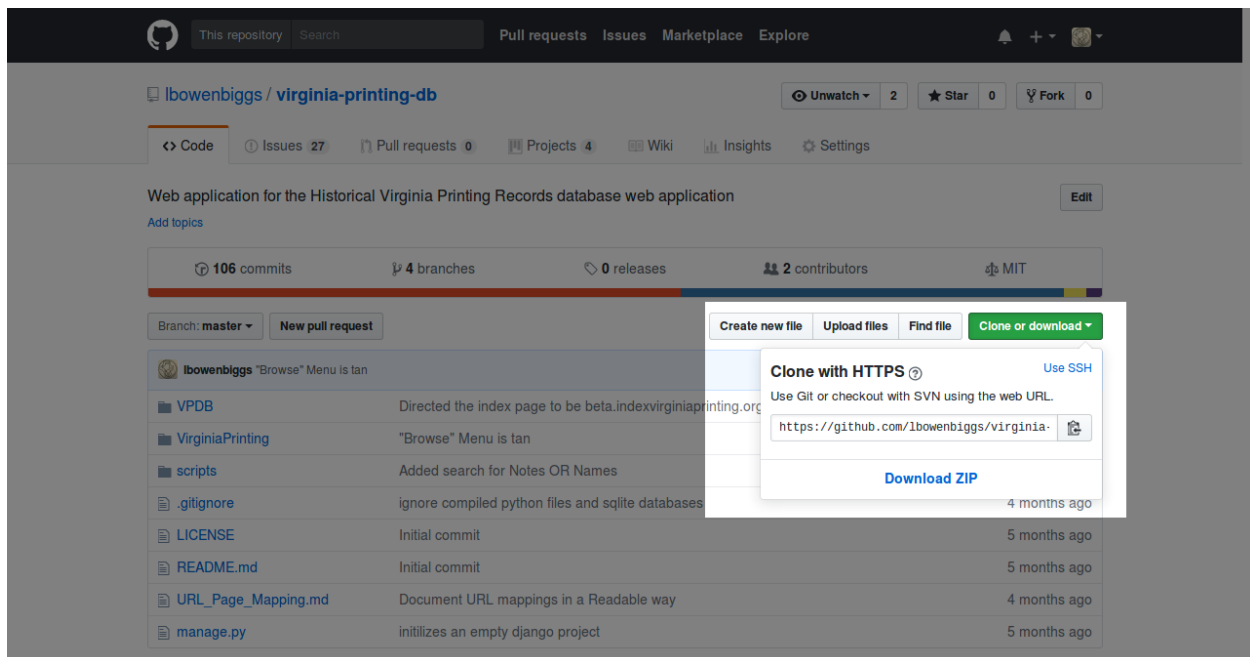


Figure 40: Download the application code as a ZIP file from GitHub.

2. Log into cPanel File Manager

Now log into the web server. On reclaim hosting, this is done by entering credentials at <https://portal.reclaimhosting.com/>. At the 'Client Area' select “cPanel” (Figure 41). Then in cPanel, click “File Manager” (Figure 42).

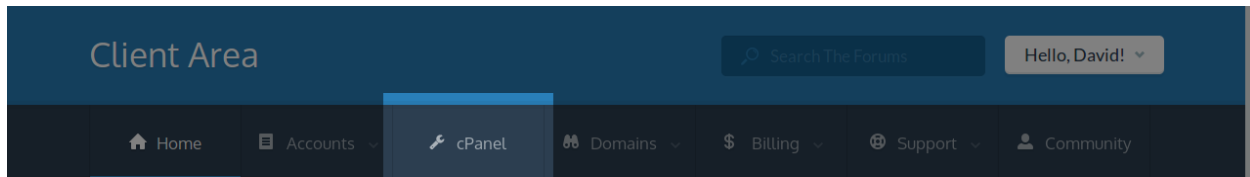


Figure 41: Click 'cPanel' on the Client Area navigation toolbar

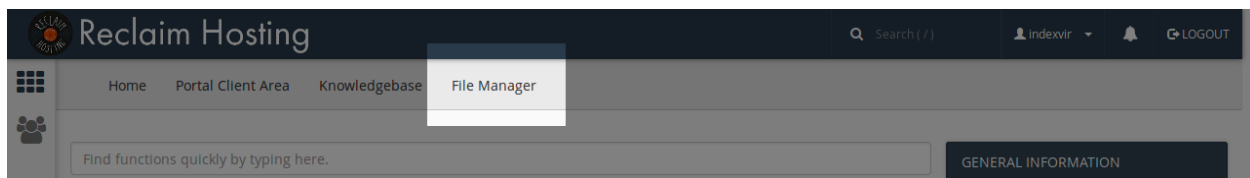


Figure 42: Click 'File Manager' in the cPanel navigation toolbar.

3. Upload the code using the file manager

With the home directory selected (the top-level folder, /home/indexvir in Figure 43), click the “Upload” button in the toolbar. On the next page, either click and drag the local file or use the “Select File” button to upload the local file. Once the file is done uploading (Figure 44), navigate back to the file manager.

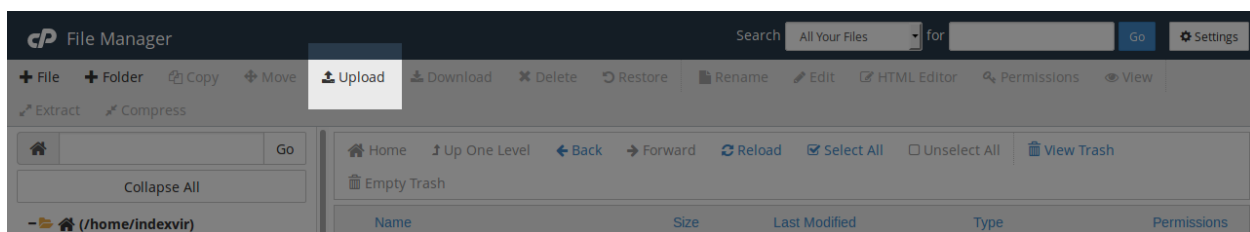


Figure 43: Select 'Upload' in the File Manager

Select the file you want to upload to “/home/indexvir”.

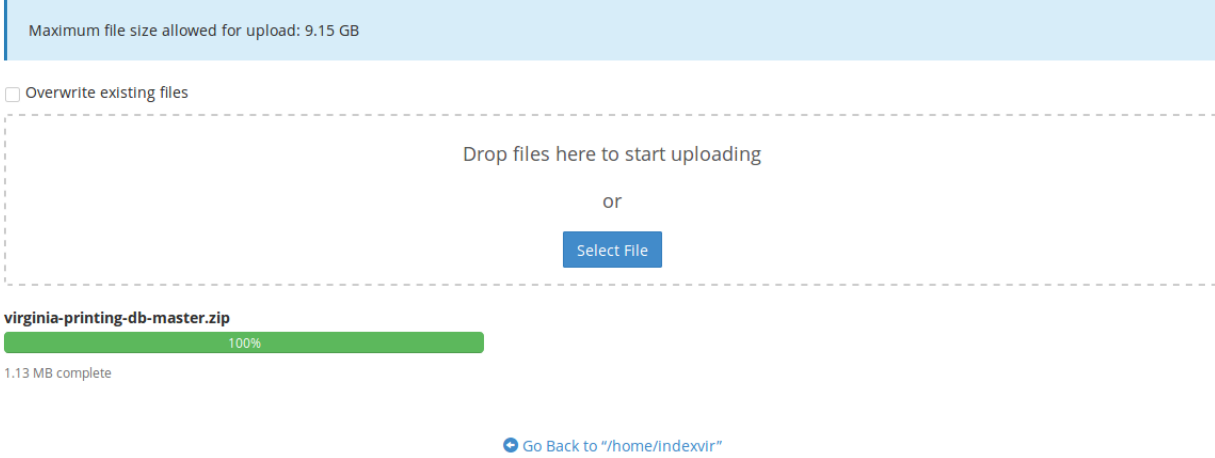


Figure 44: The ZIP file containing the code has successfully uploaded

4. Extract the Code

With the ZIP file selected, click “Extract” from the File Manager toolbar (Figure 45). In the dialog that pops up, leave the path field blank (Figure 46). The application will extract itself into a folder in the home directory named “virginia-printing-db-master” (Figure 47).

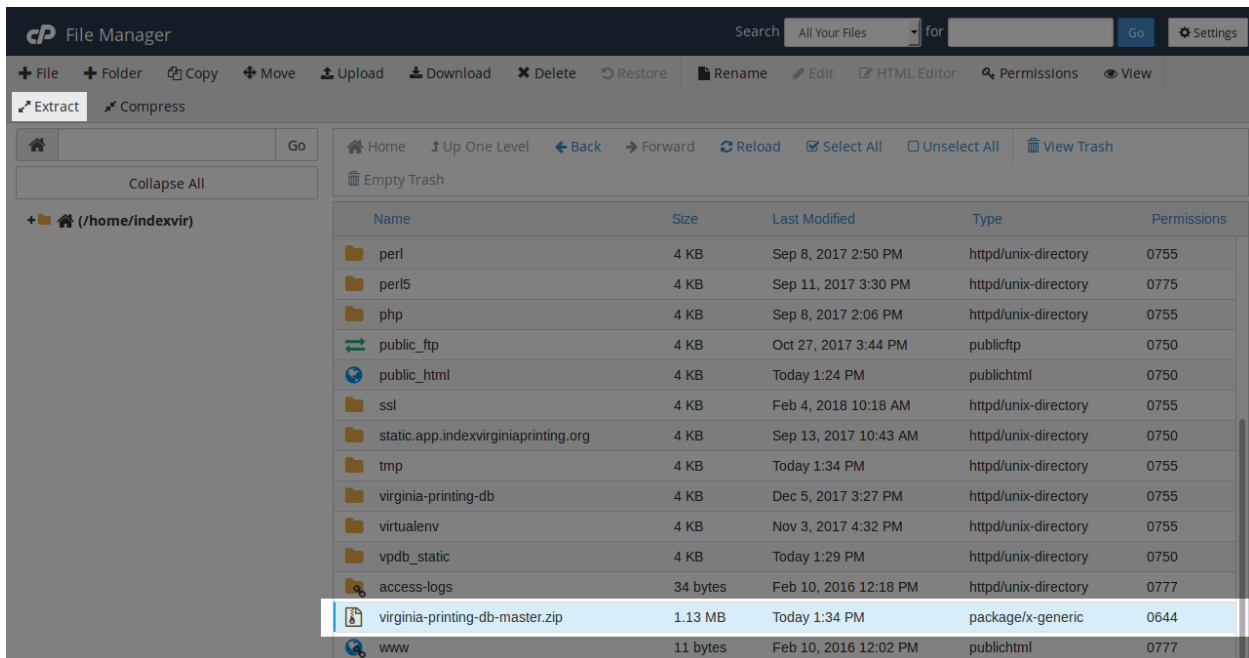


Figure 45: With the ZIP file selected, click “Extract” from the File Manager Toolbar

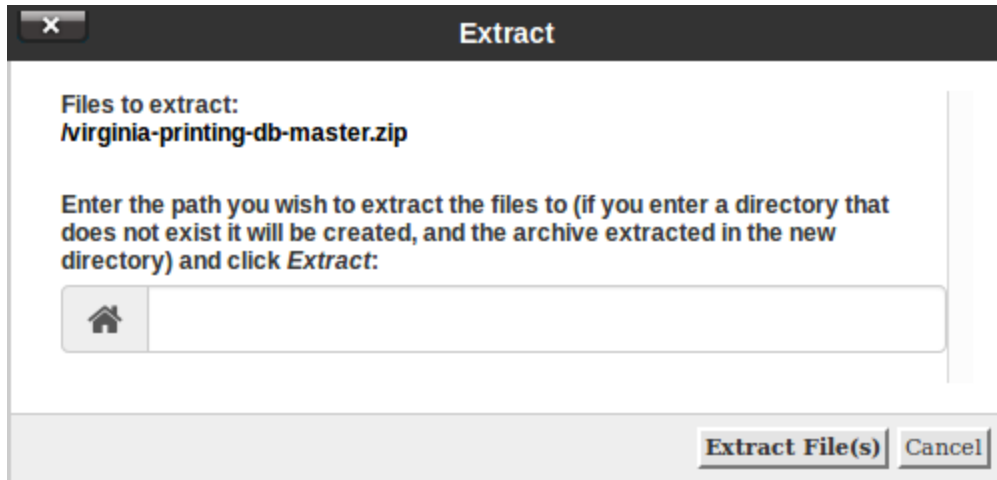


Figure 46: In the “Extract” dialog, leave the 'path you wish to extract the files to' blank.

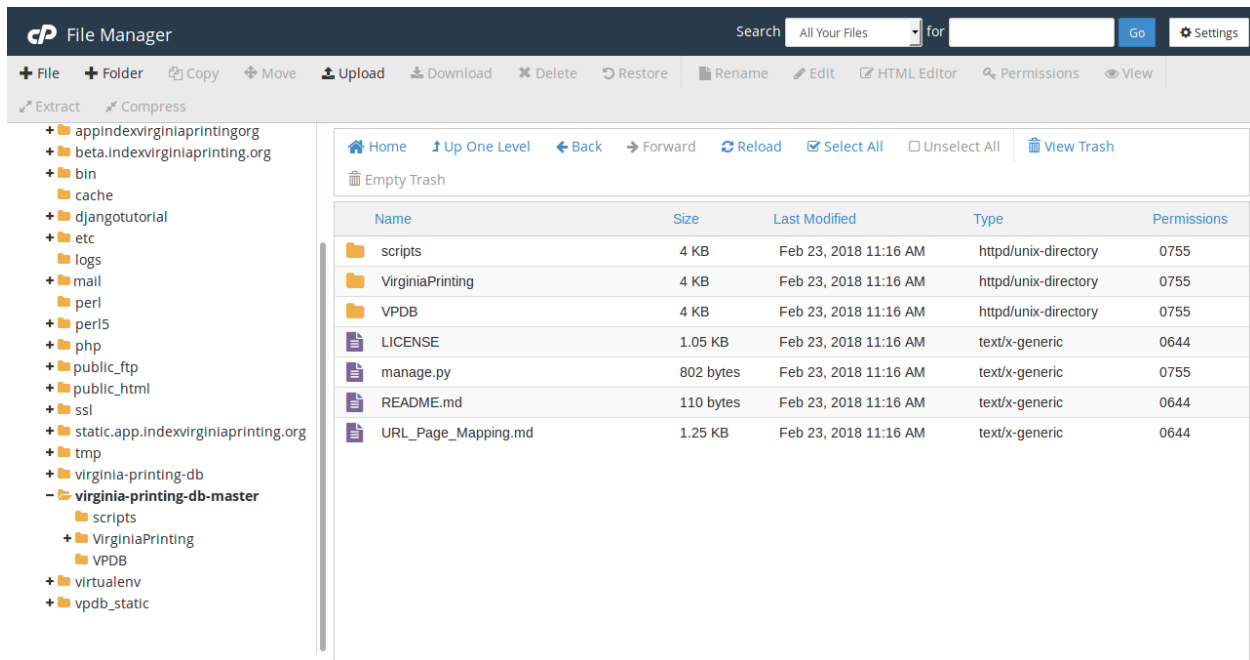


Figure 47: Application code is available on the system.

Advanced: Via git

This method is appropriate if changes are being made to the application code and the administrator wants to easily keep the application up to date. However, this method requires additional configuration to use, and is not appropriate for most administrators. In addition, to use this method you will need an SSH client such as PuTTY.

1. Create SSH Keys to access the server

This step will create a secure “key”, so the administrator can directly access the command line interface on the server. Log into the server (for example, <https://portal.reclaimhosting.com/>) and select cPanel from the Client Area toolbar (see Figure 41, above). Under the “Security” heading, select “SSH Access” (Figure 48). Click through “Manage SSH Keys” (Figure 48). Under the “Manage SSH Keys” header, click “Generate a New Key” (Figure 49).

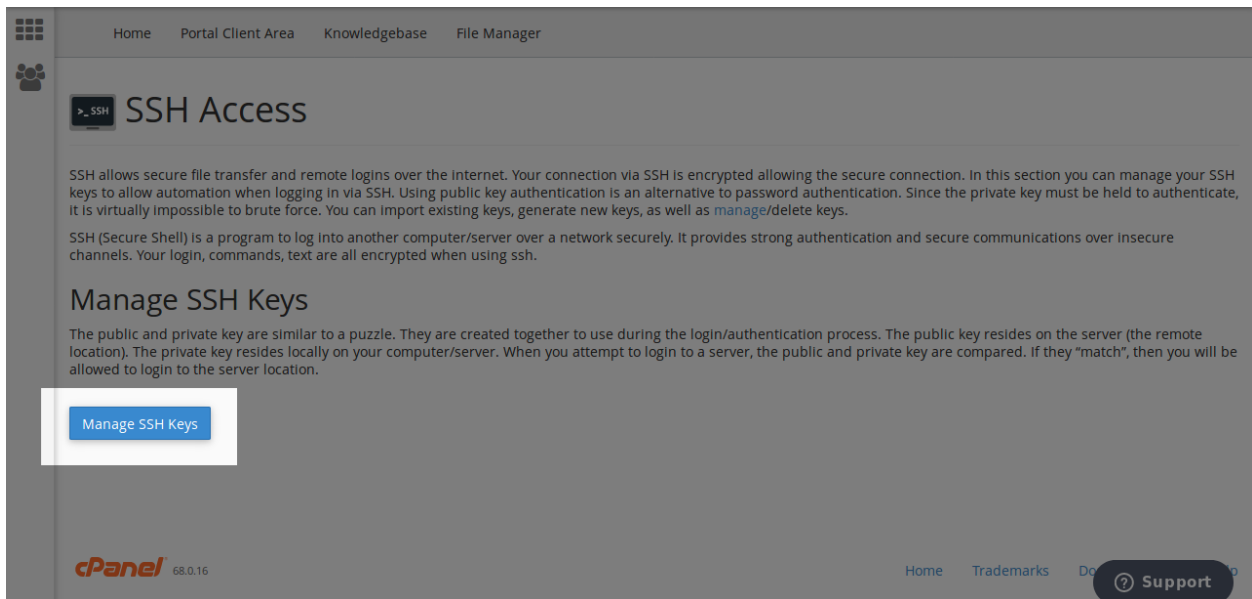


Figure 48: In the initial “SSH Access” screen, click “Manage SSH Keys”

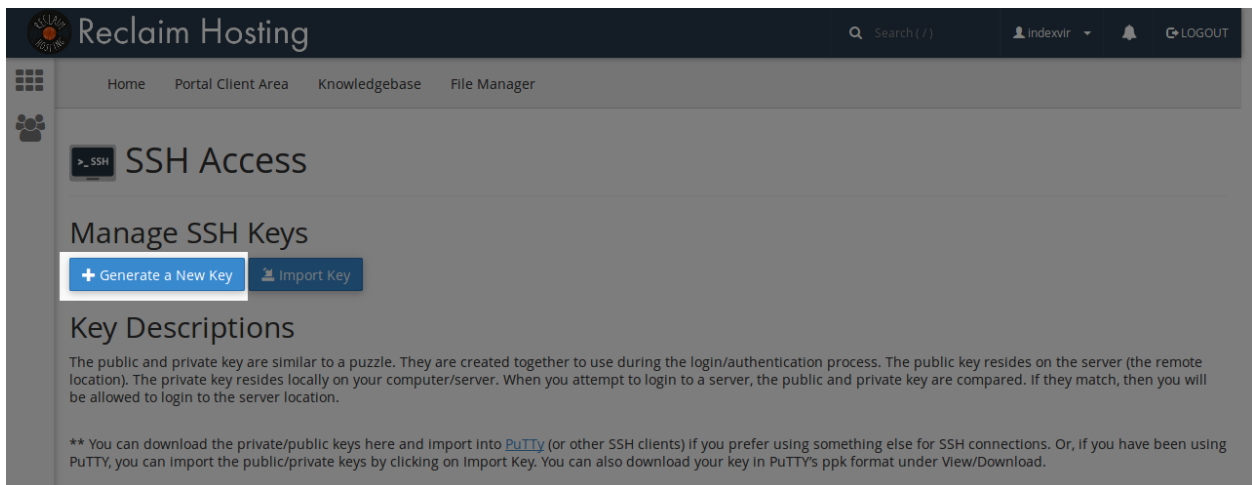


Figure 49: Under the “Manage SSH Keys” heading, click “Generate a New Key”

Leave the default settings for the key alone. The default is to set the Key Name as “id_rsa”, the Key Type as RSA, and the key size as 2048 bits. Create a strong password for the key, which will help secure it against unauthorized use (Figure 50). Click “Generate Key” to create the key. The next page will show the successful creation of the key (Figure 51).

SSH Access

Reclaim Hosting

Key Name (This value defaults to "id_rsa"):

id_rsa

Key Password:

Reenter Password:

Strength **ⓘ**

Very Weak (0/100) Password Generator

Key Type:

RSA

Key Size:

2048

Generate Key

Go Back

Figure 50: Create a strong password for the key

SSH Access

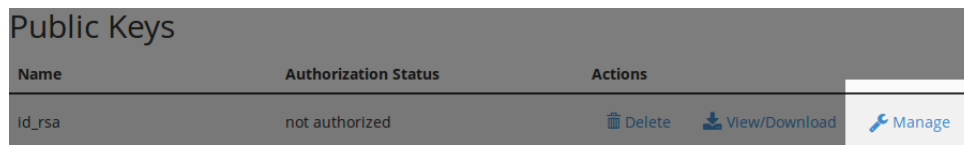
Key Generation Complete!

```
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/indexvir/.ssh/id_rsa.
Your public key has been saved in /home/indexvir/.ssh/id_rsa.pub.
The key fingerprint is:
cd:f2:88:d5:50:37:3d:34:7d:4d:98:8b:84:b3:92:5b
The key's randomart image is:
+--[ RSA 2048 ]-----+
|          ..000=0|
|         .0...=+|
|        .. + . 0.|
|       o=E . . |
|      S++       |
|     o.+        |
|    . . .       |
|-----+-----+
```

Go Back

Figure 51: The successful creation of the key

Finally, authorize the public key. This will allow it to be used to log into the server. Under the “Public Keys” heading, select “Manage” for the key that was created. Note the “Authorization Status” says that it is “not authorized” (Figure 52). In the next screen, click “Authorize” (Figure 53).



Name	Authorization Status	Actions
id_rsa	not authorized	Delete View/Download Manage

Figure 52: Click “Manage” to authorize the SSH key that was created

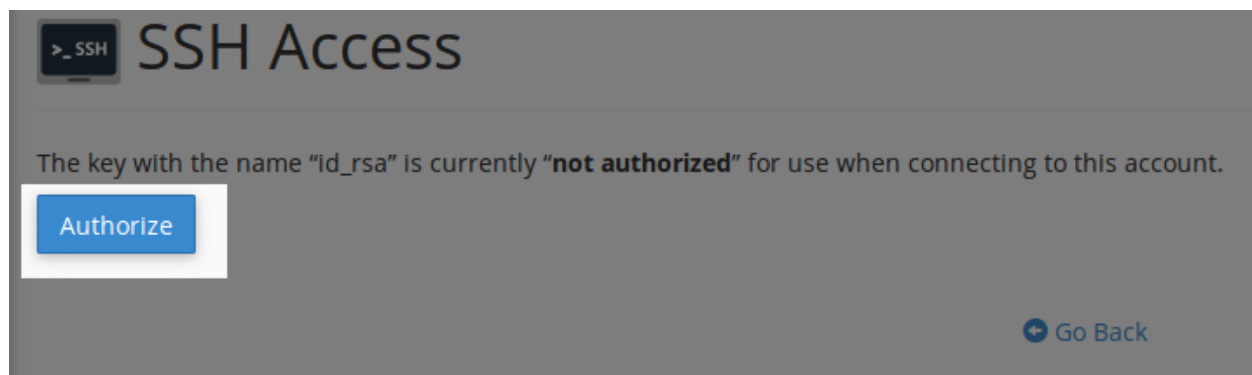


Figure 53: Click “Authorize” to authorize the SSH key for access to the account.

2. Configure a local SSH client

This tutorial assumes that the SSH client is PuTTY⁷, a free application available for Windows. A guide to configuring PuTTY can be found at [Indiana University's Website](https://kb.iu.edu/d/aews)⁸.

3. Clone the application code

Please see [GitHub's guide to cloning a repository](https://help.github.com/articles/cloning-a-repository/)⁹ for more information.

Configuring the Database

Before the application will work, it needs to be connected to a database. This section will explain how to set up and configure a MySQL database on the same server that hosts the code.

⁷ <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

⁸ <https://kb.iu.edu/d/aews>

⁹ <https://help.github.com/articles/cloning-a-repository/>

Create a MySQL Database

First the administrator of the web application must create a MySQL database.

1. Create Database

In cPanel, under the “Databases” heading, select “MySQL® Databases” (Figure 54). Then, under “Create a New Database”, give the database a name and click “Create Database” (Figure 55). In this example, the name given is 'example_db'. This name is appended to the username of the administrator, so the full name of the database is 'indexvir_example_db'.

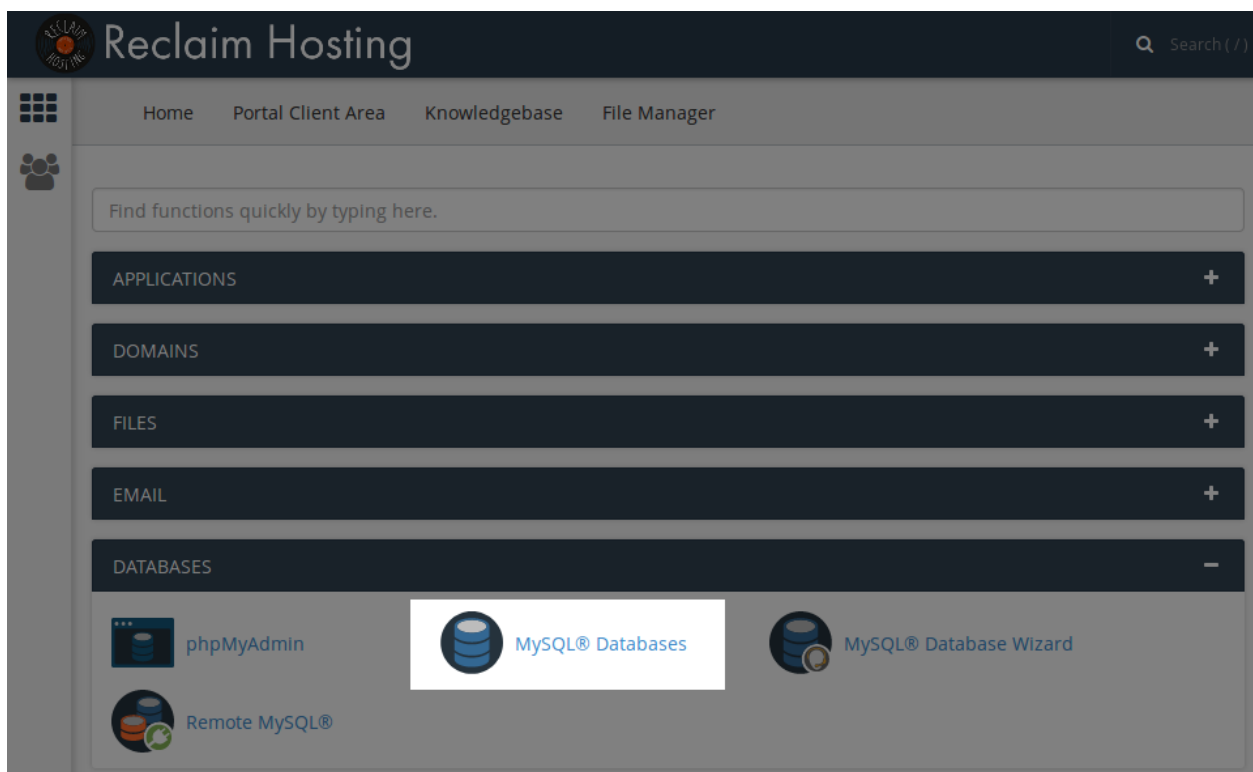


Figure 54: Under “Databases”, select “MySQL® Databases”

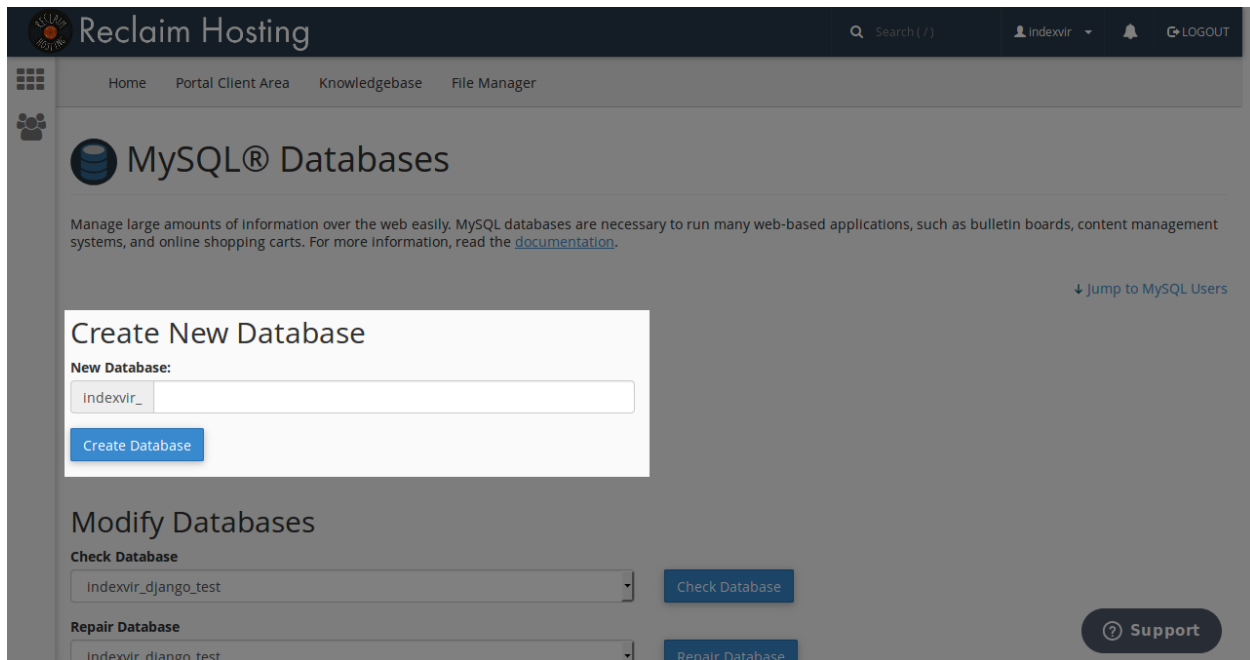


Figure 55: Create a Database by giving it a name

2. Create a User

A database user is separate from the administrative user of the server and must be created separately. Under “MySQL Users”, give the user a strong password and a name (which, like the database name, will be prefixed with the username of the server) (Figure 56). In this example, the username is 'exUser', so the full username is 'indexvir_exUser'.

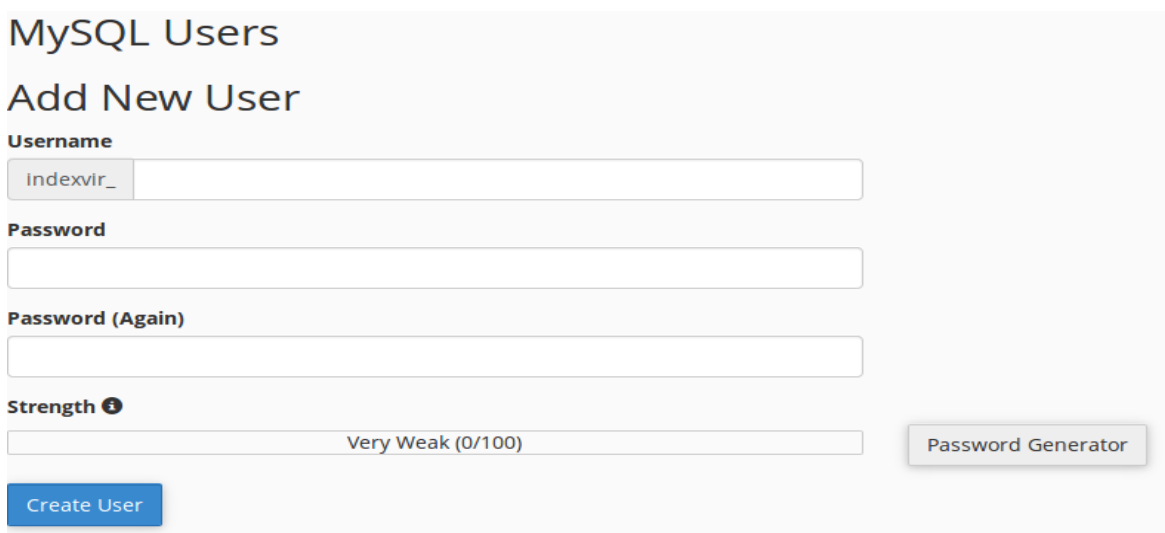
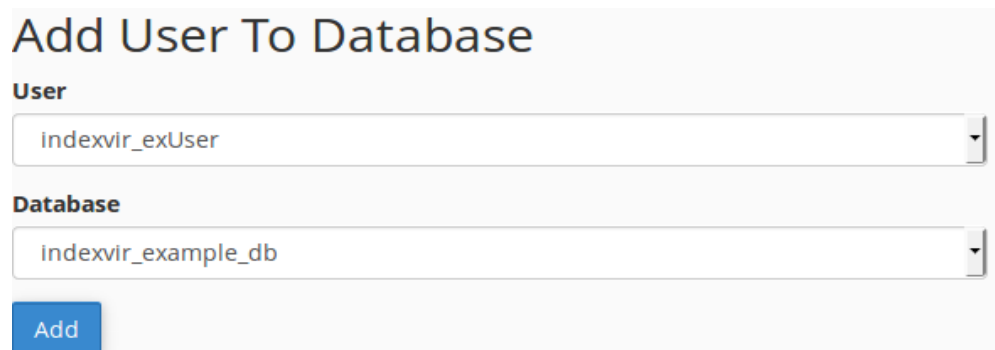


Figure 56: Create a new MySQL User with a name and strong password

3. Add User to Database

Add the user to the database that was created (Figure 57). This gives the user permission to read and write to the database. This 'user' is going to be the application, so it will need all permissions (Figure 58).



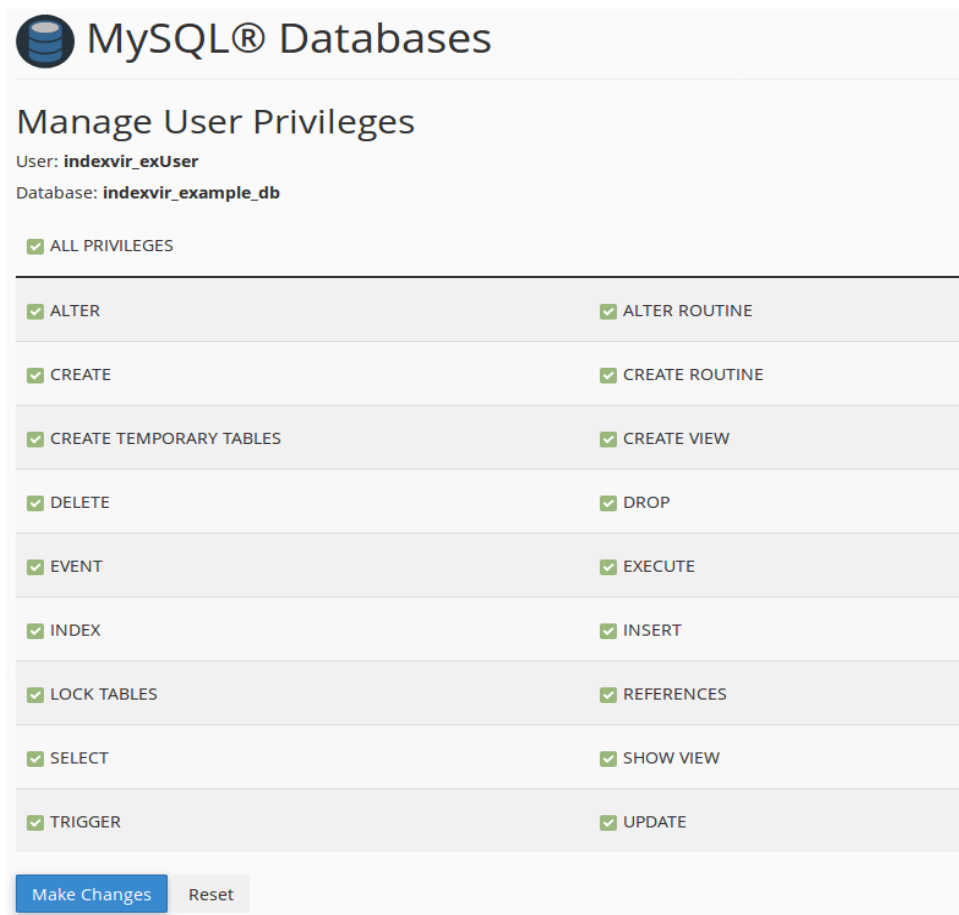
Add User To Database

User
indexvir_exUser

Database
indexvir_example_db

Add

Figure 57: Add the user to the database that was created by selecting them from the menus



MySQL® Databases

Manage User Privileges

User: **indexvir_exUser**
Database: **indexvir_example_db**

ALL PRIVILEGES

<input checked="" type="checkbox"/> ALTER	<input checked="" type="checkbox"/> ALTER ROUTINE
<input checked="" type="checkbox"/> CREATE	<input checked="" type="checkbox"/> CREATE ROUTINE
<input checked="" type="checkbox"/> CREATE TEMPORARY TABLES	<input checked="" type="checkbox"/> CREATE VIEW
<input checked="" type="checkbox"/> DELETE	<input checked="" type="checkbox"/> DROP
<input checked="" type="checkbox"/> EVENT	<input checked="" type="checkbox"/> EXECUTE
<input checked="" type="checkbox"/> INDEX	<input checked="" type="checkbox"/> INSERT
<input checked="" type="checkbox"/> LOCK TABLES	<input checked="" type="checkbox"/> REFERENCES
<input checked="" type="checkbox"/> SELECT	<input checked="" type="checkbox"/> SHOW VIEW
<input checked="" type="checkbox"/> TRIGGER	<input checked="" type="checkbox"/> UPDATE

Make Changes Reset

Figure 58: Give the MySQL user all permissions on the database

Remote Access

When populating the database with many records, it is much easier to import directly into MySQL instead of using the admin interface to individually add each one. To connect to the database, the MySQL client must be on a whitelisted domain. To find out what the local domain is, attempt a connection. The error message will include the domain name which you can then whitelist.

1. Attempt to Connect to the Database

Open MySQL Workbench and select “Connect to Database...” under “Database (Figure 59). In the dialog, enter the name of the server and database user to connect with. In this case, connect to indexvirginiaprinting.org with indexvir_exUser (Figure 60). Leave the port number and all other settings in their defaults. When you click “OK”, you will get an error message (Figure 61) that reads “Host '[hostname] [domain name]' is not allowed to connect to this MySQL server”. The domain name is what will be used in the next step to whitelist this client computer.



Figure 59: In MySQL Workbench, Connect to a Database

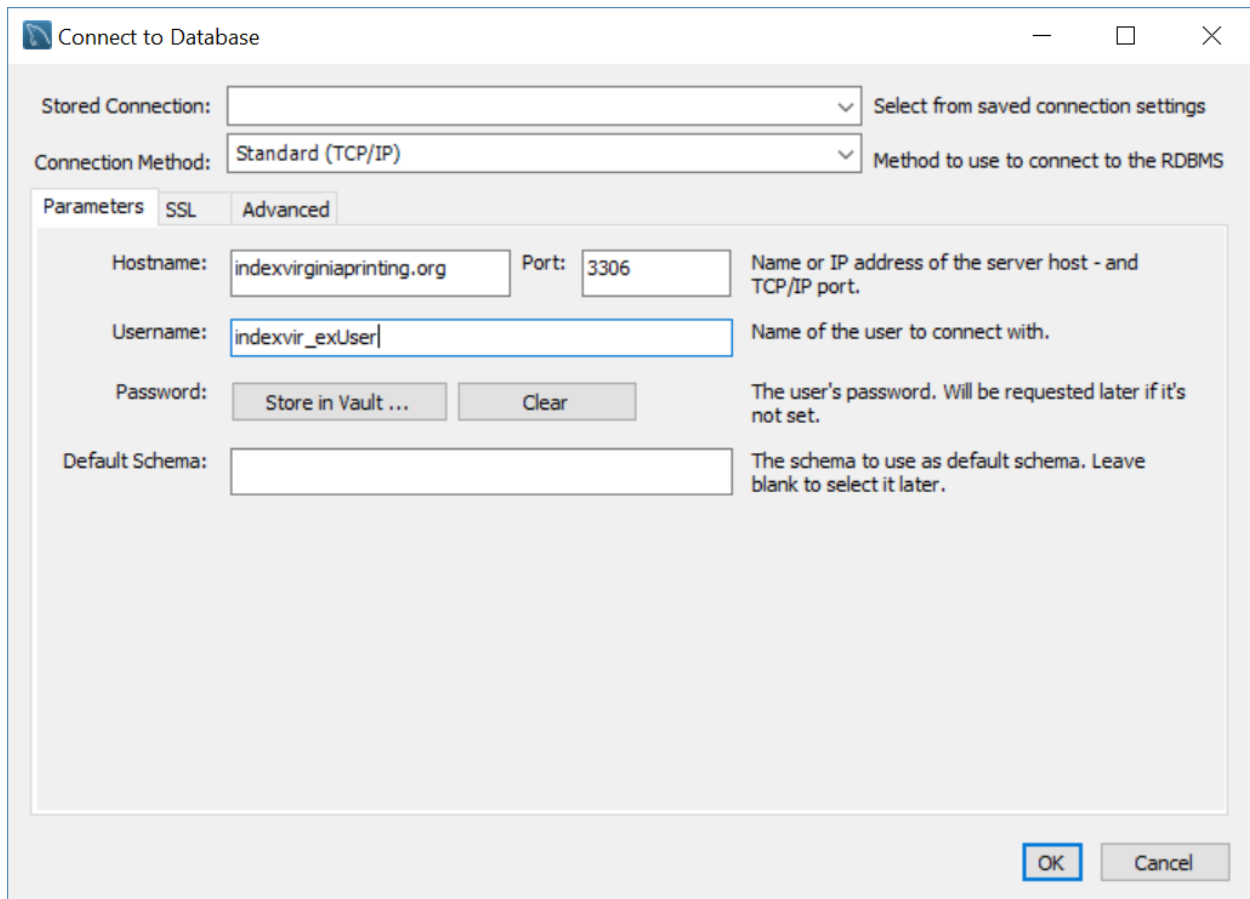


Figure 60: The “Connect to Database” dialog. Fill out Hostname and Username

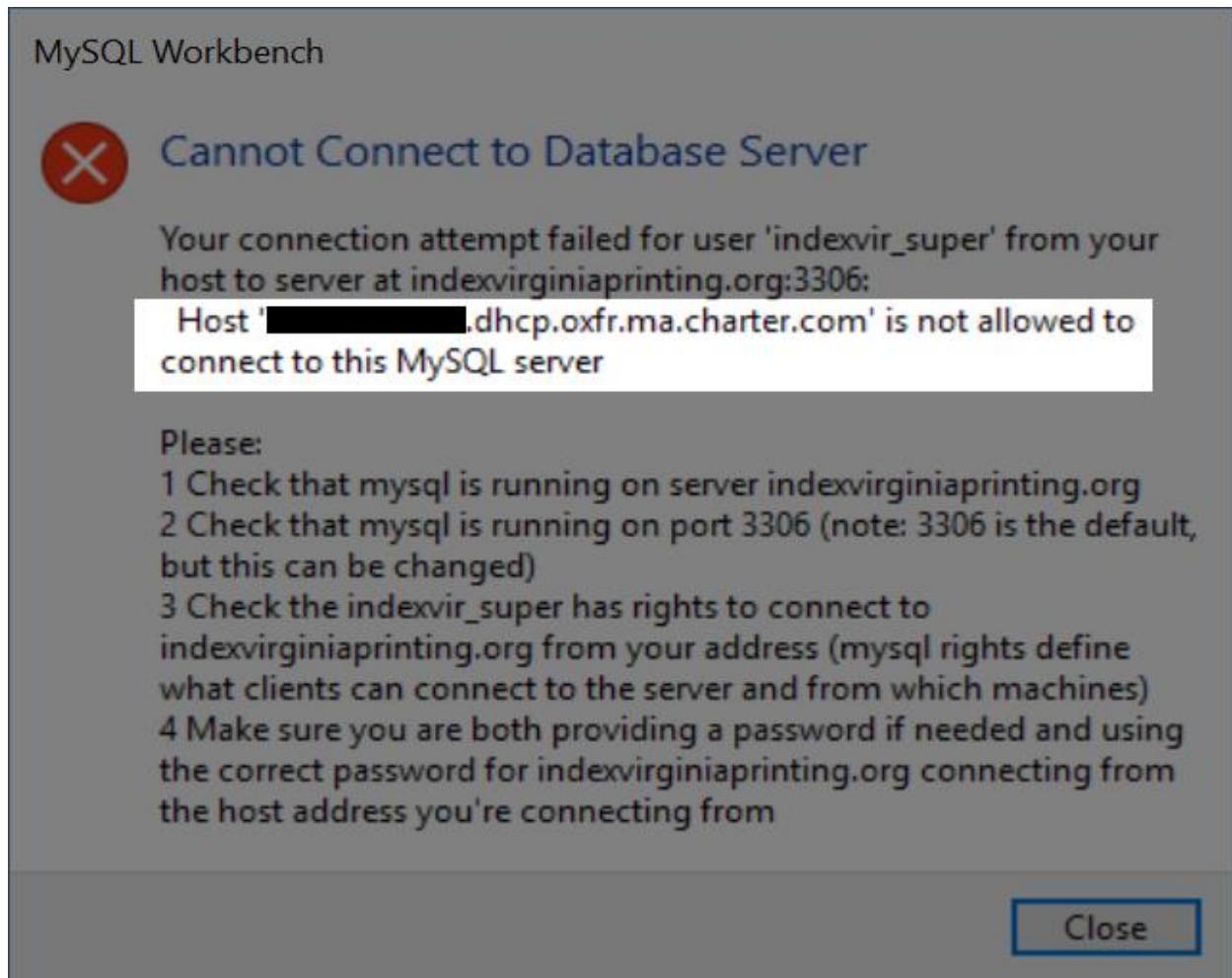


Figure 61: Error message revealing the domain name of the local machine.

2. Whitelist your Domain Name

In cPanel, under the “Databases” heading, select “Remote MySQL®” (Figure 62). Under “Add Access Host”, add the domain name of the client computer. Use the wildcard character '%' to allow any host within your network to access the database (Figure 63). Click “Add Host”

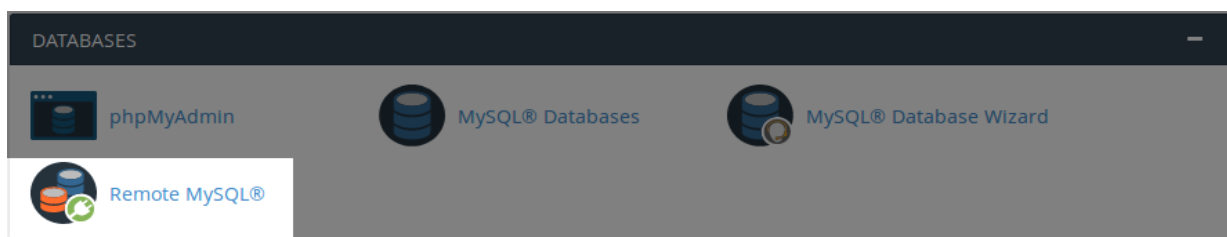


Figure 62: Under the “Databases” heading, select “Remote MySQL®”

Add Access Host

Host (% wildcard is allowed)

%.dhcp.oxfr.ma.charter.com

Add Host

Figure 63: Add the client computer's domain name to allow it to connect to the database

3. Connect to the Database

In MySQL Workbench, connect to the database as before (Figure 60, above). This time when “OK” is clicked, MySQL Workbench will prompt for the database user's password. When this is entered, it will display the database (Figure 64).

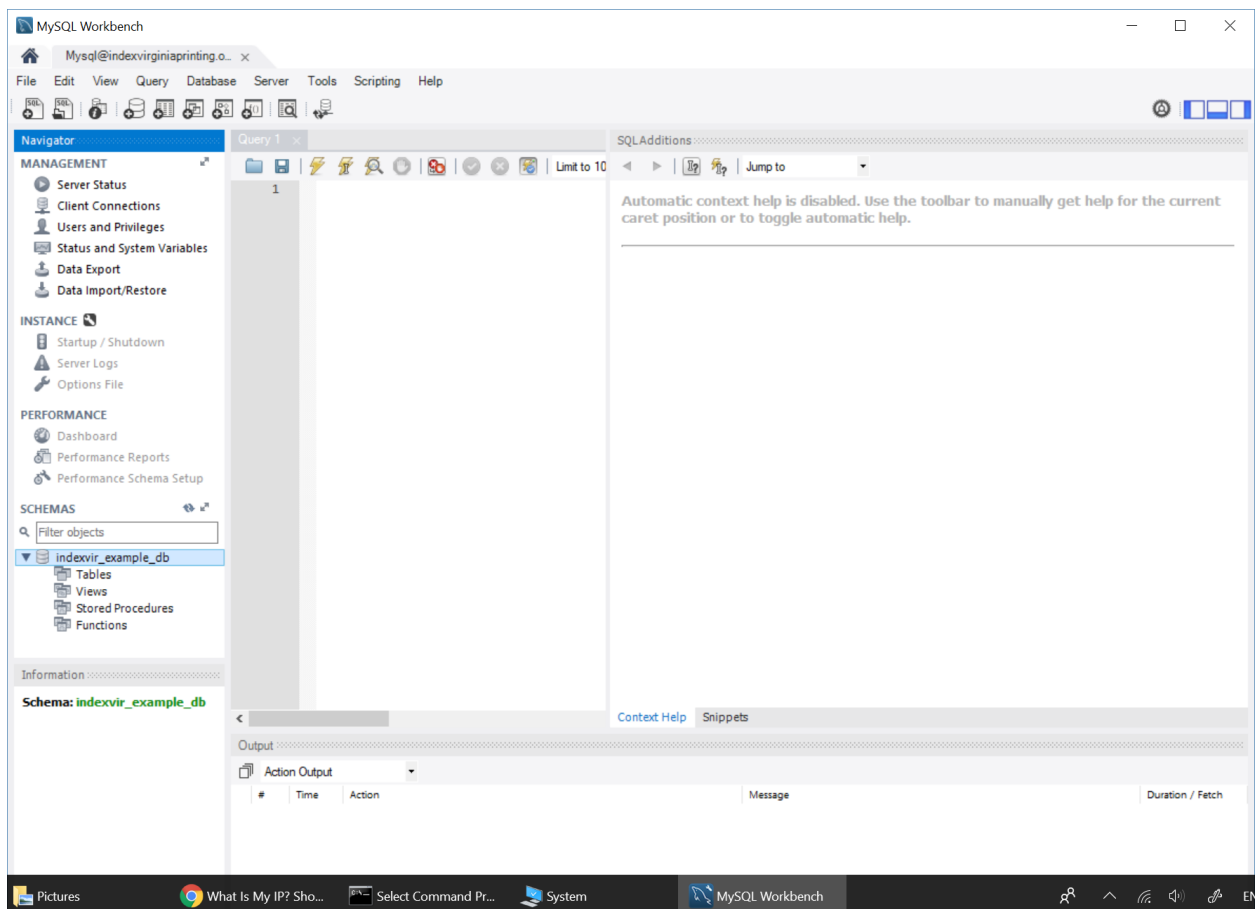


Figure 64: Successful connection between the client and the MySQL database

Configuring the Application

The next step is to configure the server to run the code that has been uploaded to it. This will require manipulating server settings through cPanel, editing the application settings file, and running a few commands.

Creating a Python Application in cPanel

In the 'software' section of cPanel, select “Setup Python App” (Figure 65).

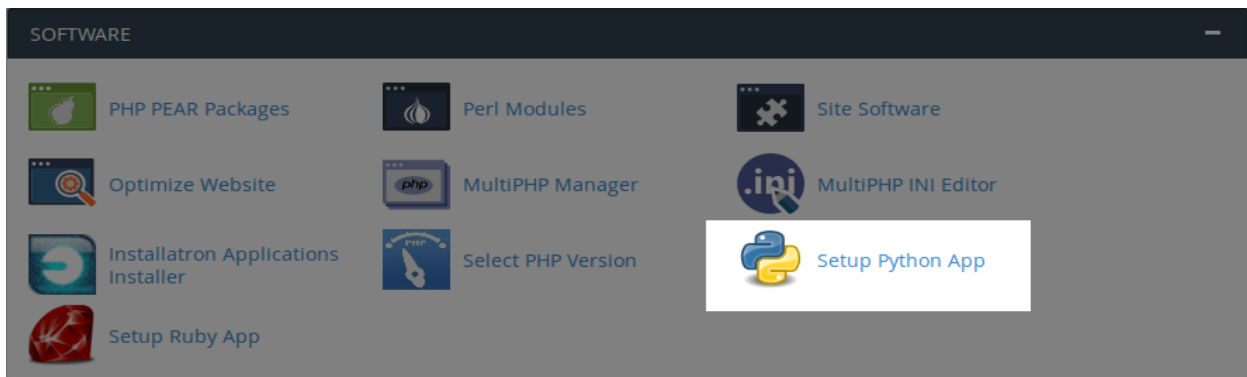


Figure 65: “Select Python App” under 'Software'

On the “Setup Python App” page, fill out the “Setup new application” form (Figure 66). For the *Index*, the Python version is 3.5. The App Directory is the folder you unzipped the application code to (see “Getting the Code”, above). Leave the App Domain/URI as the default to make the application appear on the main page of your domain. When you click “Setup”, you will see the settings for the application under “Existing applications” (Figure 67).

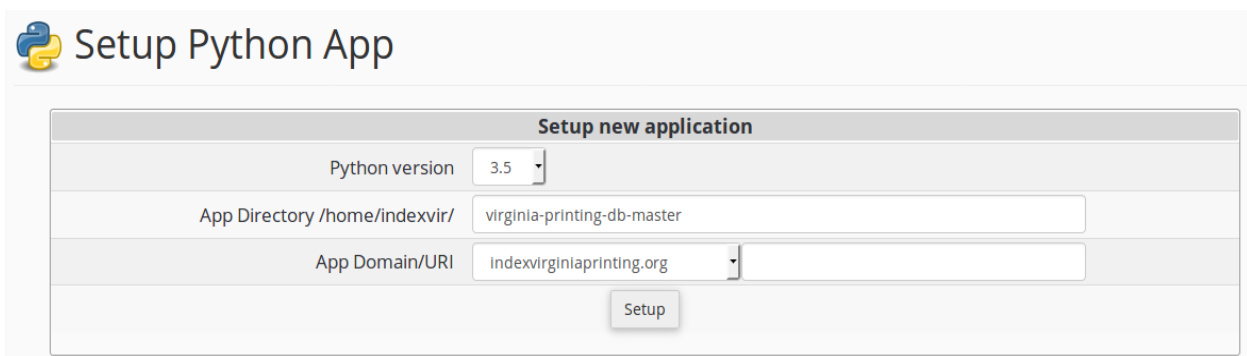
A screenshot of the 'Setup Python App' page in cPanel. The page has a light gray background and a Python logo at the top left. Below the logo is the title 'Setup Python App'. A form titled 'Setup new application' is centered on the page. The form contains three rows of input fields: 'Python version' with a dropdown menu set to '3.5', 'App Directory /home/indexvir/' with a text input field containing 'virginia-printing-db-master', and 'App Domain/URI' with a dropdown menu set to 'indexvirinlprinting.org'. A 'Setup' button is located at the bottom right of the form.

Figure 66: “Setup new application” settings for the *Index*

App Directory	virginia-printing-db-master	Edit
App URI	indexvirginiaprinting.org/	Edit
WGI file location		Edit
Python version	3.5 ▾	
modules	show	
Execute command	0-9a-zA-Z /_.,"->< symbols are allowed	<input type="button" value="Run"/>
Command for entering to virtual environment	source /home/Indexvir/virtualenv/virginia-printing-db-master/3.5/bin/activate	
<input type="button" value="Update"/> <input type="button" value="Reset"/> <input type="button" value="Restart"/> <input type="button" value="Remove"/>		

Figure 67: Settings for the new application under “existing applications”

Edit application settings

Next, you will need to edit the python application settings. First, click “Edit” on the far right of the “WGI file location” row. Enter “VPDB/wsgi.py:application” and click “Save” (Figure 68).

This tells the server where to start the application.

App Directory	virginia-printing-db-master	Edit
App URI	indexvirginiaprinting.org/	Edit
WGI file location	VPDB/wsgi.py:application	Save
Python version	3.5 ▾	
modules	show	
Execute command	0-9a-zA-Z /_.,"->< symbols are allowed	<input type="button" value="Run"/>
Command for entering to virtual environment	source /home/Indexvir/virtualenv/virginia-printing-db-master/3.5/bin/activate	
<input type="button" value="Update"/> <input type="button" value="Reset"/> <input type="button" value="Restart"/> <input type="button" value="Remove"/>		

Figure 68: Edit the WGI file location in the application settings

To add dependencies, click “Show” in the “modules” column. In the text box, type “Django” (It may be slow to load as it searches through the modules). Select the “Django” dependency, and then version 1.11.5, which is what version this application uses. Click “Add”. Now add the dependency for “mysqlclient” and click “Add”. At the bottom of the application settings, click “Update”. You will see that it successfully loaded the modules (Figure 69). A few other modules (pytz and wheel, for example) may be added as well.

App Directory	virginia-printing-db-master	Edit
App URI	indexvirginiaprinting.org/	Edit
WGI file location	VPDB/wsgi.py:application	Edit
Python version	3.5 ▾	
modules	<input type="text"/> Add	
	Django 1.11.5 [x]	
	mysqlclient 1.3.12 [x]	
	pytz 2018.3 [x]	
	wheel 0.30.0 [x]	
	hide	
Execute command	<input -><="" allowed"="" are="" symbols="" type="text" value="0-9a-zA-Z /_-,\"/> Run	
Command for entering to virtual environment	source /home/indexvir/virtualenv/virginia-printing-db-master/3.5/bin/activate	
Update Reset Restart Remove		

Figure 69: Completed application settings for the *Index*.

Modify settings.py for Production

The code is, by default, for development. You will need to edit a few things to make it suitable for production. In the File Manager, navigate to “virginia-printing-db-master/VPDB/settings.py” and click the “Edit” button (Figure 70). This will take you to a text editor (Figure 71). Make the sure the following settings are correct:

- Line 26: Debug must be set to False to prevent the server from giving details that may compromise the security of the application.

DEBUG = False

- Line 28: Allowed Hosts must include the URL of the website, so the application can run

ALLOWED_HOSTS = ['indexvirginiaprinting.org']

- Lines 80-89: The Database must be configured to use the MySQL database created above.

Set the NAME property to be the name of the database you created, the USER property to be the user of the database you created and update the PASSWORD to match that user. Ensure the ENGINE is set to use MySQL and the HOST and PORT are set to the location of the server and the port MySQL is running on (3306 is the default).

```
DATABASES = {
    'default': {
```

```

'ENGINE': 'django.db.backends.mysql',
'NAME': 'indexvir_production',
'USER': 'indexvir_admin',
'PASSWORD': '',
'HOST': 'indexviriniaprinting.org',
'PORT': '3306'
}
}

```

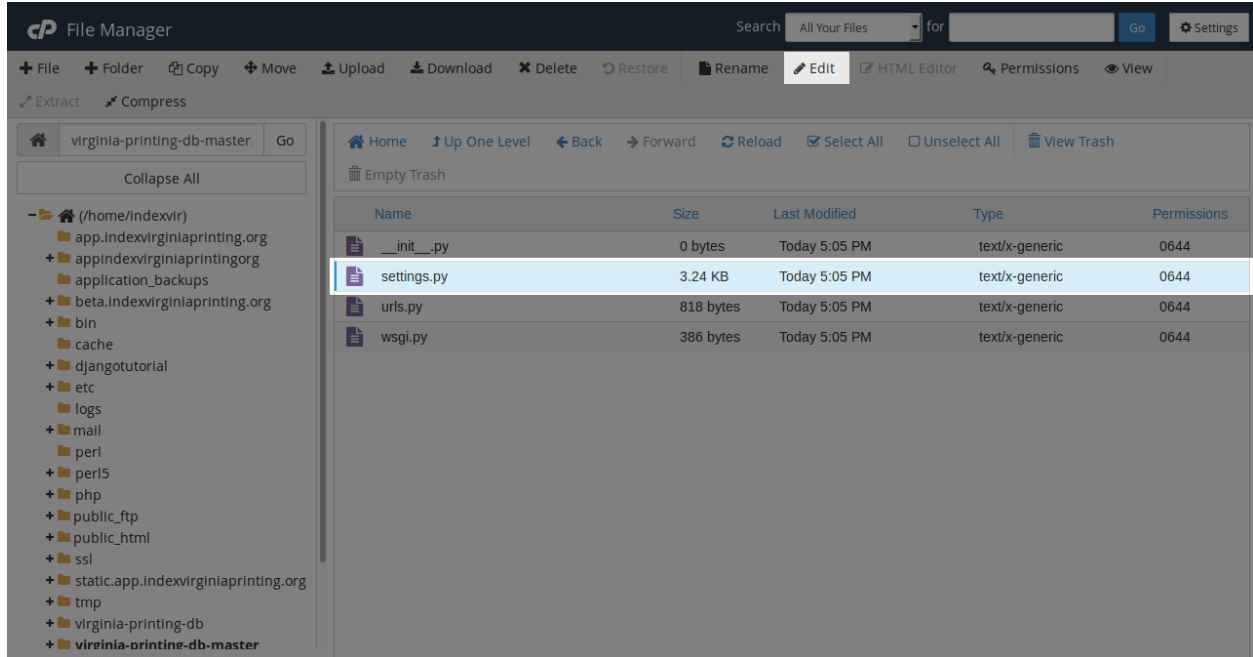


Figure 70: Select “Edit” to update the settings for production

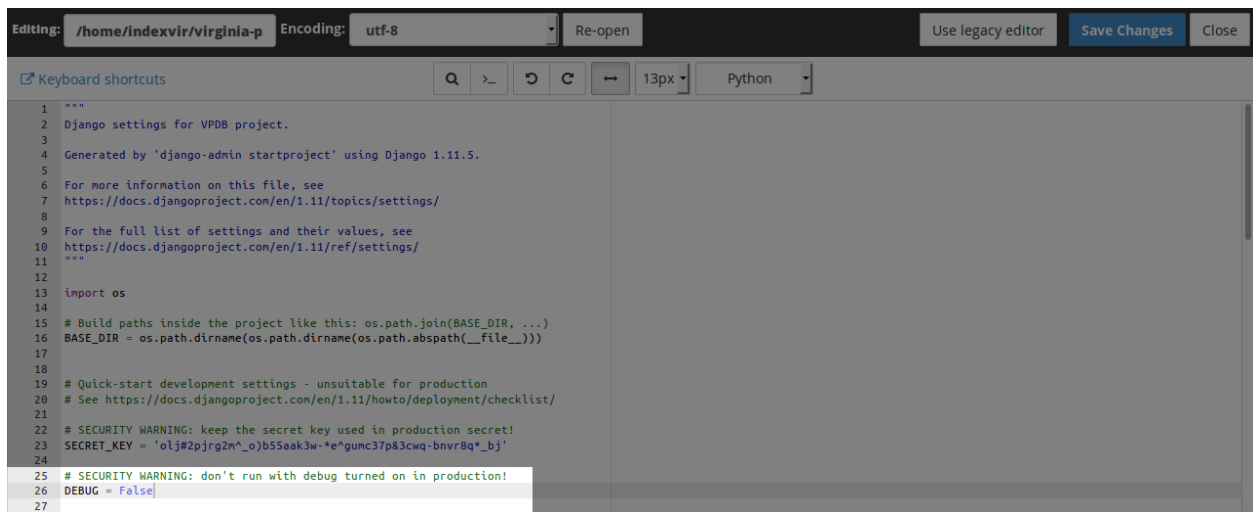


Figure 71: Change the Debug settings through the text editor on the server.

Publishing the Application

The application is almost ready to use! To finalize the application, you will need to configure static files and upload data to the server, so it can be accessed through the application.

Configure Static Files

Some of the application files are “static”, meaning they don't depend on the contents of the database. These files are handled directly by the server, not through the application. For performance reasons, these files should be separate from the application.

1. Create a Subdomain

The static files will live on a subdomain of the main website at `static.indexvirginiaprinting.org`. To create the subdomain, click “Subdomains” under the “Domains” heading in cPanel (Figure 72). Then name the subdomain “static” and let the Document Root auto populate. Click create (Figure 73).

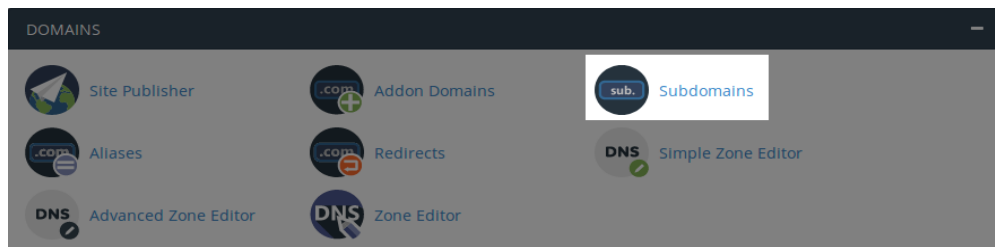


Figure 72: Create a subdomain through cPanel

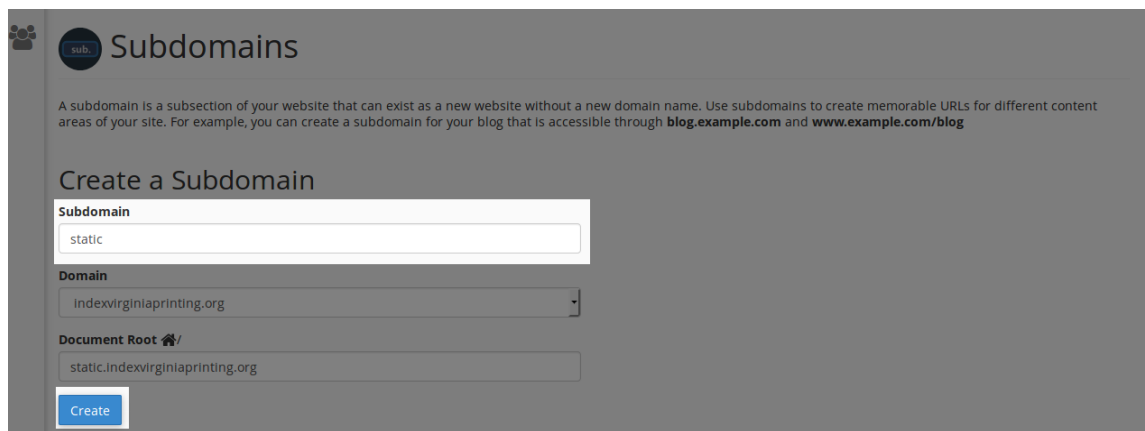


Figure 73: Name the subdomain 'static' and use the default document root.

2. Configure STATIC_URL & STATIC_ROOT

Next, change the settings of the application so it knows where to find the static files. Open settings.py for editing (see “Modify settings.py for Production”, above). And change the static settings to the following:

```
STATIC_URL = 'http://static.indexvirginiaprinting.org/'  
STATIC_ROOT = '/home/indexvir/static.indexvirginiaprinting.org/'
```

The STATIC_URL should be set to the subdomain created in the previous step. The STATIC_ROOT should be set to the Document Root created for the subdomain in the previous step. Save settings.py.

3. Run collectstatic

Now, the application must move the static files into the static directory. From the “Setup Python App” page (see “Edit Application Settings”, above), enter the following into the 'Execute Command' Row (Figure 74):

```
./manage.py collectstatic --no-input
```

This will run the collectstatic command on the remote server and copy the files to the appropriate location. You can verify the files copied correctly by using cPanel's file explorer (Figure 75).

App Directory	virginia-printing-db-master	Edit
App URI	indexvirginiaprinting.org/	Edit
WSGI file location	VPDB/wsgi.py:application	Edit
Python version	3.5 ▾	
modules	show	
Execute command	<input type="text" value="/manage.py collectstatic --no-input"/>	<input type="button" value="Run"/>
Command for entering to virtual environment	source /home/Indexvir/virtualenv/virginia-printing-db-master/3.5/bin/activate	
<input type="button" value="Update"/> <input type="button" value="Reset"/> <input type="button" value="Restart"/> <input type="button" value="Remove"/>		

Figure 74: Run the collectstatic command from the python application settings page

Home ↑ Up One Level ← Back → Forward ↻ Reload <input checked="" type="checkbox"/> Select All <input type="checkbox"/> Unselect All 🗑 View Trash				
🗑 Empty Trash				
Name	Size	Last Modified	Type	Permissions
📁 records	4 KB	Today 6:52 PM	httpd/unix-directory	0775
📁 refs	4 KB	Today 6:52 PM	httpd/unix-directory	0775
🖼 background-2136168_960_720.jpg	230.8 KB	Today 6:52 PM	image/x-generic	0664
📄 checkboxes.js	2.57 KB	Today 6:52 PM	text/x-generic	0664
📄 default.css	1.79 KB	Today 6:52 PM	text/css	0664
🖼 richmond1800banner.jpg	20.32 KB	Today 6:52 PM	image/x-generic	0664
🖼 richmondprogress1800.jpg	518.8 KB	Today 6:52 PM	image/x-generic	0664

Figure 75: Verify the collectstatic command ran properly by checking the contents of `static.indexvirginiaprinting.org/VirginiaPrinting/`

4. Add documents to the static directory

The application expects PDF versions of the records, which it links to as printable PDFs. Upload the appropriate documents to `static.indexvirginiaprinting.org/VirginiaPrinting/records` under the relevant directory.

Setting up the Database

Django sets up the database based on the record types specified in the application. To set up the database with the right tables, run the following command from the Python Application Settings 'Execute command' row (Figure 76):

```
./manage.py migrate
```

App Directory	virginia-printing-db-master	Edit
App URI	indexvirginiaprinting.org/	Edit
WSGI file location	VPDB/wsgi.py:application	Edit
Python version	3.5 ▾	
modules	show	
Execute command	<input type="text" value="./manage.py migrate"/>	<input type="button" value="Run"/>
Command for entering to virtual environment	source /home/Indexvir/virtualenv/virginia-printing-db-master/3.5/bin/activate	
<input type="button" value="Update"/> <input type="button" value="Reset"/> <input type="button" value="Restart"/> <input type="button" value="Remove"/>		

Figure 76: Run the migrate command to configure the database tables

You can verify that the tables were added by logging into the database through MySQL Workbench and viewing the list of tables (Figure 77).

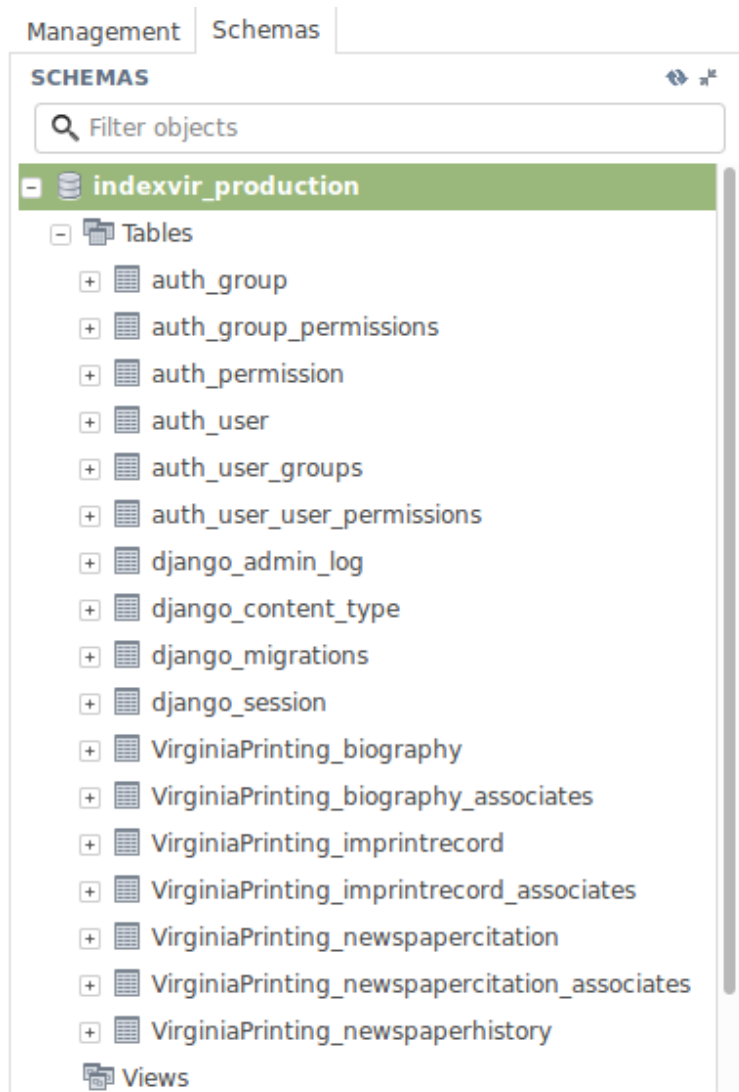


Figure 77: The tables created by Django for the application in MySQL Workbench

Populate the Database

Now you will populate the database with the actual historical data. For each of the 7 tables in the Virginia Printing application (these tables all start with 'VirginiaPrinting_'), you will need to:

1. Run the 'Table Data Import Wizard'
2. Select file to import
3. Import the file into the table

For details on this process, see the “Table Data Import Wizard”¹⁰ section of the [MySQL Workbench documentation](#). Once the database is populated, you will be able to see the application live at <https://indexvirginiaprinting.org/>

Adding the first Admin User

Finally, you must create an administrator account for the application, so they may log into the administrator website and modify records. To do this, log into the server via SSH (See “Configure a Local SSH Client”, above). Navigate to the application directory and run `createsuperuser`. Provide a username, email, and password¹¹ (Figure 78). These cannot be recovered if you forget them. Once the admin is successfully created, you will be able to log into the [administrator website](#)¹² with the credentials you just created (Figure 79).

```
(virginia-printing-db-master:3.5)indexvir@unwound [~/virginia-printing-db-master]# ./manage.py createsuperuser
Username (leave blank to use 'indexvir'): admin
Email address:
Password:
Password (again):
Superuser created successfully.
```

Figure 78: Creating a superuser on the command line

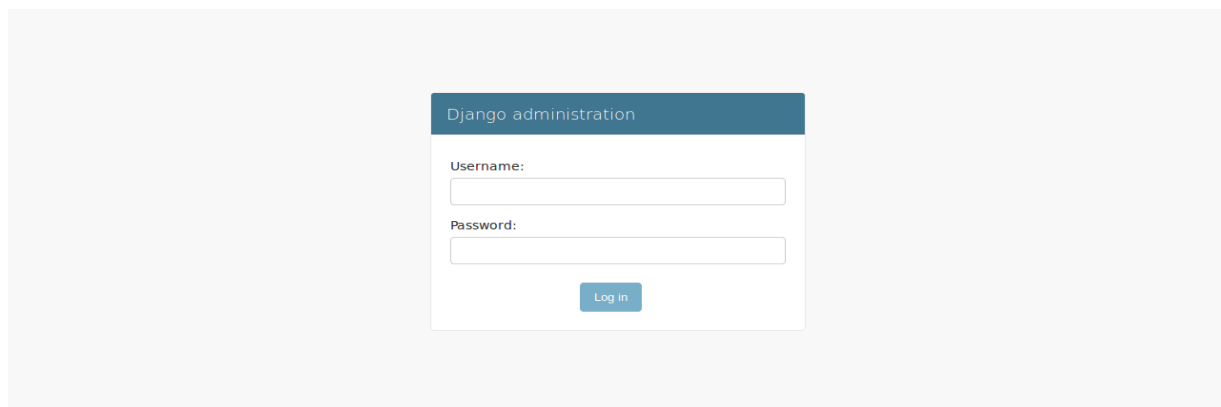


Figure 79: Administrator login page

¹⁰ <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-table.html>

¹¹ For more information on creating a superuser, see the Django documentation at <https://docs.djangoproject.com/en/1.11/intro/tutorial02/#introducing-the-django-admin>

¹² Available at <https://indexvirginiaprinting.org/admin/>